Contents

OPERATING SYSTEMS (IV-SEM.)

UNIT 1

INTRODUCTION TO OPERATING SYSTEMS – FUNCTION, EVOLUTION, DIFFERENT TYPES, DESIRABLE CHARACTERISTICS AND FEATURES OF AN O/S

Q.1. What is an operating system ? Discuss the difficulties involved in writing an operating system for a real-time environment. Give examples.
(R.G.P.V., June 2009)

Ans. An operating system may be viewed as an organized collection of software extensions of hardware, consisting of control routines for operating a computer and for providing an environment for execution of programs. Other programs rely on facilities provided by the operating system to gain access to computer system resources, such as files and input/output (I/O) devices. Programs usually invoke services of the operating system by means of ***operating-system calls.*** In either case, the operating system acts as interface between users and hardware of a computer system.

Difficulties Involved in Writing an Operating System for a Real-time Environment – The main difficulties are keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time, it could cause a breakdown of the entire system it is running. Therefore, when writing an operating system for a real-time system, the writer must be sure that his scheduling schemes do not allow response time to exceed the time constraint.

Q.2. What is bare machine ? ***(R.G.P.V., Dec. 2015)***

Ans. Bare machine means a computer without its operating system. After the development of programmable computers but prior to the development of operating systems, programs were fed to the computer system directly using machine language by the programmers without any system software support. This approach is termed as the bare machine approach in the development of operating systems. Here, there is no memory management. We give the program

all of the memory, with no limitation. This provides maximum flexibility to the user, and minimum hardware cost.

Q.3. Write the name of system components of OS. (R.G.P.V., Dec. 2014)

Ans. An operating system is an important part of each computer system. A computer system can be divided into four components – the hardware, the operating system, the application programs, and the users. The hardware – the central processing unit (CPU), the memory, and the input/output (I/O) devices – provides the basic computing resources. The application programs such as word processors, spreadsheets, compilers, and web browsers–define the ways in which these resources are used to solve the computing problems of the users. The operating system controls and coordinates the use of the hardware among various application programs for the various users.

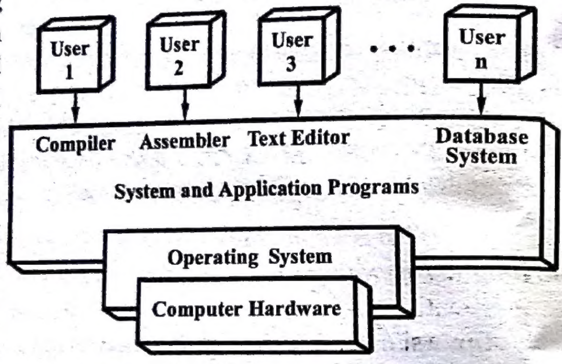


Fig. 1.1 Abstract View of the Components of Computer System

Q.4. What are the three main purposes of an operating system ? (R.G.P.V., Dec. 2014)

Ans. The main purposes of an operating system are as follows –

(i) To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.

(ii) To allocate the separate resources of the computer as needed to solve the problem given.

(iii) As a control program it serves two major functions –

(a) Supervision of the execution of user program to prevent errors and improper use of the computer and

(b) Management of the operation and control of I/O devices.

Q.5. What are the two main functions of an operating system ? (R.G.P.V., June 2016)

Or

What are the main functions of an operating system ? (R.G.P.V., Dec. 2011)

Or

Explain the functions of operating system. (R.G.P.V., Dec. 2013)

Ans. An operating systems perform two basically unrelated functions, extending the machine and managing resources –

(i) The Operating System as an Extended Machine – The function of the operating system is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.

(ii) The Operating System as a Resource Manager – Internally an operating system acts as a manager of resources of the computer system such as processor, memory, files, and I/O devices. In this role, the operating system keeps track of the status of each resource, and decides who gets a resource, for how long, and when. In systems that support concurrent execution of programs, the operating system resolves conflicting requests for resources in a manner that preserves system integrity, and in so doing attempts to optimize the resulting performance.

Q.6. Write short note on evolution of operaing systems.

Ans. An operating system is the system process its workload serially or concurrently. It means the resources of the computer system can only be shared by a single program, until it is completed or resources can be dynamically reassigned among a collection of active programs in different stages of execution. These type of operating systems are known as multiprogramming systems due to Their ability to execute the multiple programs in interleaved fashion. Both the serial and multiprogrammed operating systems having several variations. In particular, we describe batch processing, multiprogramming and serial processing in operating systems. Mainly the focus is on the progression of some ideas to develop the operating system, rather to attach the dates on the chart of the history. Moreover, the historical development of operating systems for mainframes was retraced by minicomputers and then by microcomputers and personal computers. Thus many concepts and ideas are recurring and applicable to different systems at different times.

Thus the program execution technique and a brief sketch of the process of program development, namely, edit-compile-execute cycle, are described for each type of operating system. Though program execution is an essential activity in all the computer systems, and for the program-development environments the productivity of program preparation is important.

Q.7. Discuss different structures of operating system with advantages and disadvantages. (R.G.P.V., June 2010, Dec. 2015)

Ans. There are four types of operating system structures –

(i) Monolithic systems (ii) Layered systems

(iii) Virtual machines (iv) Client-server model.

(i) Monolithic Systems – This approach might well be subtitled "The Big Mess". The structure is that there is no structure. The operating system is written as a collection of procedures, each of which can call any ones whenever it needs to. When this technique is used, each procedure in the system has a well-defined interface in terms of parameters and results, and each one is free to call any other one.

However, in monolithic systems, it is possible to have at least a little structure. The services (system calls) provided by the operating system are requested by putting the parameters in well-defined place, such as in registers or on the stack, and then executing a special trap instruction known as a ***kernel call*** or ***supervisor call***.

This instruction switches the machine from user mode to kernel mode (also called supervisor mode), and transfers control to the operating system, shown as event (1) in fig. 1.2. The operating system then examines the parameters of the call to determine which system call is to be carried out, shown as (2) in fig. 1.2. Next, the operating system indexes into a table that contains in slot k a pointer to the procedure that carries out system call k. This operation, shown as (3) in fig. 1.2, identifies the service procedure which is to be called. Finally, the system call is finished and control is given back to the user program.

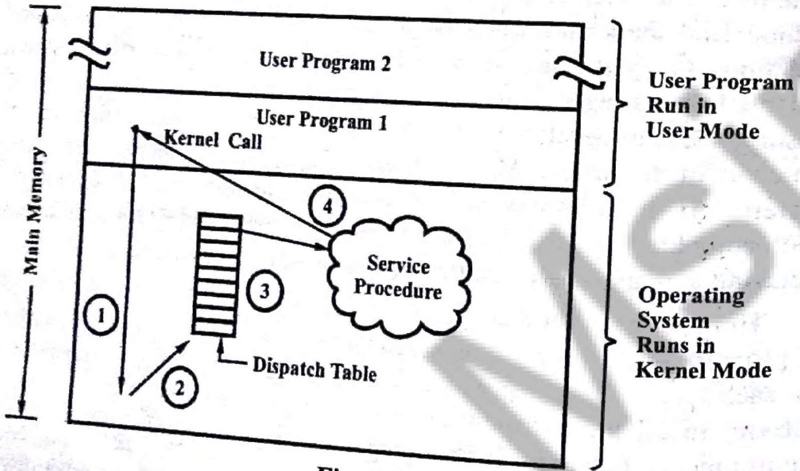


Fig. 1.2

This organization suggests a basic structure for the operating system –

(a) A main program that invokes the requested service procedure.

(b) A set of service procedures that carry out the system calls.

(c) A set of utility procedures that help the service procedures.

The division of the procedures into three layers is shown in fig. 1.3.

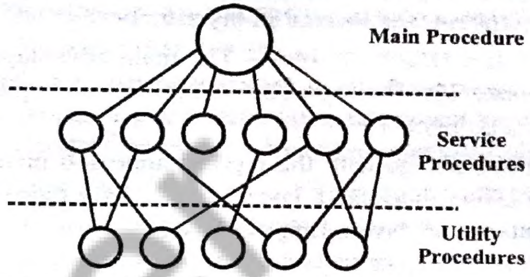


Fig. 1.3 A Simple Structuring Model for a Monolithic System

Advantages and Disadvantages – Since, there are few interfaces between the application programs and the underlying hardware, monolithic operating systems deliver a good performance. But on the other hand, it is difficult to enhance or maintain such an operating system and modifications to one module of the operating system can affect other components modules.

(ii) Layered Systems – The approach for the structure of Monolithic System was generalized to organize the operating system as a hierarchy of layers, each one constructed upon the one below it. The first system constructed using this way was THE system built at the Technische Hogeschool Eindhoven in the Netherlands by E.W. Dijkstra in 1968.

The system has 6 layers, as shown in fig. 1.4.

Layer	Function
5	The operator
4	User programs
3	Input / output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Fig. 1.4 Structure of the 'THE' Operating System

Layer 0 dealt with allocation of the processor, switching between processes when interrupts occurred or timers expired. In other words, layer 0 provided the basic multiprogramming of the CPU.

Layer 1 did the memory management. It allocated space for processes in main memory and on a 512 K word drum used for holding parts of processes (pages) for which there was no room in main memory.

Layer 2 handled communication between each process and the operator console. Above this layer each process had its own operator console. Layer 3 took care of managing the I/O devices and buffering the information streams to and from them. Above layer 3 each process could deal with abstract I/O devices, instead of real devices. Layer 4 contains the user programs. They did

not have to worry about process, memory, console, or I/O management. The system operator process was located in layer 5.

Advantages and Disadvantages – The main advantage of the layered approach is *modularity*. This approach simplifies debugging and system verification.

The main disadvantage with the layered approach involves the careful definition of the layers, because a layer can use only those layers below it. Another disadvantage with layered approach is that they tend to be less efficient than others.

(iii) Virtual Machines – The basic thing of virtual machine is that, it runs the program on bare hardware and does multiprogramming, and provides several virtual machine to the next layer up. Unlike all other operating systems, the virtual machines are not extended machines. Instead, they are exact copies of the bare hardware, including I/O, kernel/user mode, interrupts, and all everything else the real machine has.

There are two variants on this design. In first, one MS-DOS itself is loaded into the virtual 8086's address space, so the monitor of virtual machine just reflects the trap back to the MS-DOS. When MS-DOS tries to do the I/O itself later, that operation is caught and carried out by the virtual machine monitor.

In the second one the virtual machine monitor catches the first trap and does the I/O itself, because it knows that what all the MS-DOS system calls and what each trap does.

Advantages and Disadvantages – There are two primary advantages for using virtual machines. First, by completely protecting system resources, the virtual machine provides a robust level of security. Second, the virtual machine allows system development to be done without disrupting normal system operation.

A major disadvantage with the virtual machine approach involves disk systems.

(iv) Client-Server Model – This is the modern operating system structure, and in this structure the processes are run in oftenly user mode instead of kernel mode, but kernel handles the user requests. Basically kernel handles the communication between clients and servers. Request for service,

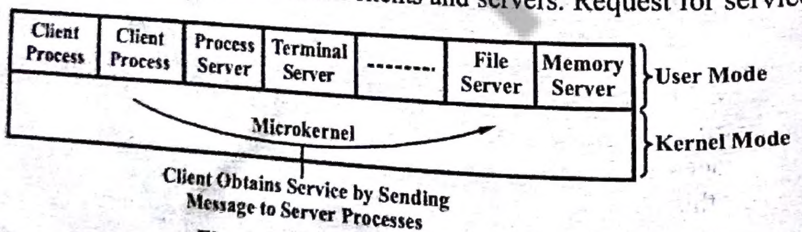


Fig. 1.5 The Client-server Model

is a user process known as ***client process***, and reply to the user is known as ***server process***. The client-server model is shown in fig. 1.5 in which a client process requests for a file, that is in the server.

In this model, the operating system is splitted up into parts. Each part handles only one facet of the system, such as process service, file service, memory service or terminal service. Each part becomes small and worth manage.

Advantages and Disadvantages – The main advantage of client-server model is its adaptability to use in distributed systems.

A disadvantage of this approach is that multi-process architecture makes inefficient use of system resources such as memory.

Q.8. Compare the monolithic and layered operating system.
(R.G.P.V., Dec. 2011)

Ans. Refer Q.7(i) and (ii).

Q.9. What are the various types of operating systems ? Discuss any one.

Ans. The various types of operating systems are as follows –

(i) Desktop system	(ii) Multiprocessor system
(iii) Clustered system	(iv) Handheld system
(v) Real-time system	(vi) Multitasking
(vii) Multiprogramming	(viii) Distributed system.

Desktop System – Personal computers PCs introduced in the 1970s. During their first decade, the CPUs in PCs lacked the features needed to protect an operating system from use programs. Therefore, PC operating systems were neither multiuser nor multitasking. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems include PCs running Microsoft Windows and the Apple Macintosh. The MS-DOS operating system from Microsoft has been superseded by multiple flavors of Microsoft Windows, and IBM has upgraded MS-DOS to the OS/2 multitasking system. The Apple Macintosh operating system has been ported to more advanced hardware, and now includes new features, such as virtual memory and multitasking. Linux, a UNIX-like operating system available for PCs, has become popular recently.

Q.10. What do you mean by multiprocessor systems ?
(R.G.P.V., Dec. 2008)

Ans. Multiprocessing or multiprocessor operating systems manage the operation of computer systems that incorporate multiple processors. Multiprocessor operating systems are multitasking systems by defintion because they support simultaneous execution of multiple tasks on different processors. In other words, a computer system that contains two or more CPUs is called a ***multiprocessor system***. It is also known as ***parallel systems***

or *tightly coupled systems.* These processors share the computer bus, the clock, memory and peripheral devices. For example, Univoc 1100/94, Cyber ITO, IBM 3081 (1980), IBM 3084.

The most common multiprocessor systems use symmetric multiprocessing (SMP), in which each processor runs an identical copy of the operating system, and these copies communicate with one another as needed. Fig.1.6 shows a SMP architecture. Some multiprocessor systems use asymmetric multiprocessing, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. Hence, this scheme defines a master-slave relationship.

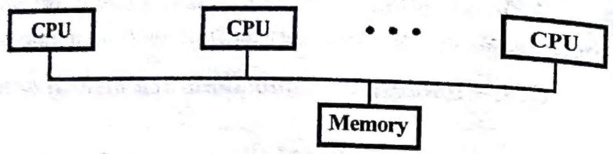


Fig. 1.6 Symmetric Multiprocessing Architecture

Q.11. What is meant by clustered systems ?

Or

Define essential properties of the clustered operating system.

(R.G.P.V., Dec. 2010)

Ans. Clustered systems gather together multiple CPUs to accomplish computational work. However, clustered systems differ from parallel systems in that they are composed of two or more individual systems coupled together. The general definition of the term clustered is that clustered computers share storage and are closely linked via LAN networking.

Clustering is usually performed to provide high availability. In asymmetric clustering, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server. In symmetric mode, two or more hosts are running applications, and they are monitoring each other.

Other forms of clusters include parallel clusters and clustering over a WAN. Parallel clusters allow multiple hosts to access the same data on the shared storage.

Q.12. Discuss about handheld systems in brief.

Or

Define essential properties of the handheld operating system.

(R.G.P.V., Dec. 2010)

Ans. Handheld systems include personal digital assistants (PDAs), such as palmpilots or cellular telephones with connectivity to a network such as the Internet. Developers of handheld systems and applications face many challenges due to the limited size of such devices.

Many handheld systems have memory between 512 kB and 8 MB. Hence, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.

Another issue of concern to developers of handheld devices is the speed of processor used in the device. Usually, processors for handheld devices run at a fraction of the speed of a processor in a PC. To achieve a faster processor in a handheld device requires a larger battery that would have to be replaced or recharged more frequently. But to minimize the size of handheld devices, smaller and slower processors are required which consume less power.

Q.13. Explain in detail about real-time systems.

Or

What is the difference between a hard real-time system and a soft real-time system ? (R.G.P.V., Dec. 2014)

Or

Compare hard real-time systems and soft real-time systems.

(R.G.P.V., Dec. 2012)

Or

What are real-time operating systems ? How are they developed and implemented ? Illustrate some applications where they can be used.

(R.G.P.V., June 2011)

Or

What do you understand by real-time operating system ? How it is different from other operating system ? (R.G.P.V., June 2016)

Ans. Real-time operating systems are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines. Such applications are industrial control, telephone switching equipment, flight control, and real-time simulations.

The real-time operating systems can be of two types – hard real-time operating system and soft real-time operating system.

(i) Hard Real-time Operating System – These operating systems guarantee that critical tasks be completed within a certain range of time. For example, a robot is hired to weld a car body, if robot welds too early or too late, the car cannot be sold , so it is a hard real-time system that require to complete car welding by robot hardly on the time.

(ii) Soft Real-time Operating System – This operating system provides some relaxation in time limit. For example – Multimedia systems, digital audio system etc.

Explicit, programmer-defined and controlled processes are encountered in real-time systems. A separate process is charged with handling a single

external event. The process is activated upon occurrence of the related event signaled by an interrupt. Multitasking operation is accomplished by scheduling processes for execution independently of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services. The processor is allocated to the highest-priority process. Higher-priority processes preempt execution of the lower-priority processes. This type of scheduling, called, priority-based preemptive scheduling is used by real-time systems.

Q.14. Explain the term time-sharing. ***(R.G.P.V., Dec. 2008)***

Ans. Multiprogrammed, batched systems provided an environment where the various system resources were utilized effectively, but it did not provide for user interaction with the computer system. Time sharing is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them but the switches occur so frequently that the users can interact with each program while it is running.

A time-shared operating system allows many users to share the computer simultaneously. Each action or command in a time-shared system tends to be short, so only a little CPU time is needed for each user. As the system switches rapidly from one user to the next, each user is given the impression that the entire computer system is dedicated to her use, however it is being shared among many users.

A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared computer. Each user has at least one separate program in memory. A program loaded into memory and executing is called as a ***process.*** When a process executes, it executes for only a short time before it either finishes or needs to perform I/O.

Time-sharing operating systems are more complex than multiprogrammed operating systems. In both, several jobs must be kept simultaneously in memory, so the system must have memory management and protection. To get a good response time, jobs may have to be swapped in and out of main memory to the disk that now serves as a backing store for main memory. A common method for achieving this goal is ***virtual memory,*** which is a technique that allows the execution of a job that may not be completely in memory.

Q.15. Define the essential properties of the time-sharing system. ***(R.G.P.V., Dec. 2010)***

Ans. Time-sharing uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal and output is normally printed immediately to the screen.

Q.16. Define the essential properties of the following types of OS –
(i) Batch (ii) Real-time (iii) Time-sharing.
(R.G.P.V., June 2011)

Or

What are the major differences between the following types of operating systems –
(i) Batch system (ii) Real-time system (iii) Time-sharing system ?
(R.G.P.V., Dec. 2011)

Or

How a time-sharing operating system differs from batch operating system ?
(R.G.P.V., Dec. 2015)

Ans. (i) Batch – Jobs with similar needs were batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is improved by attempting to keep CPU and I/O devices busy at all times through buffering, spooling, multiprogramming and off-line operating. Batch OS is good for executing large jobs that needs little interaction, it can be submitted and picked up later. Payroll system, bank statements are the examples of batch based operating system.

(ii) Real-time – Real-time OS is used to control a dedicated application. The system reads information from sensors and must respond within a certain amount of time to ensure correct performance.

(iii) Time-sharing – Uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal and output is normally printed immediately to the screen.

Q.17. What is operating system ? Define the essential properties of the following types of operting system –
(i) Batch (ii) Time-sharing
(R.G.P.V., Dec. 2016)

Ans. **Operating System** – Refer Q.1.

The essential properties of the following operating system are –

(i) Batch – Refer Q.16 (i).

(ii) Time-sharing – Refer Q.16 (iii).

Q.18. Write some characteristics of batch processing system.

Ans. The characteristics of batch processing system are as follows –

(i) Some movement of a large number of files from one device to another to prepare data for data mining.

(ii) The entry of orders for a business' products that have been mailed in.

(iii) The printing of standard reports or statements.

(iv) The unattended running of a program or programs.

(v) The collection of program input data collected over some period of time.

(vi) The output from running the programs is not needed immediately.

Q.19. Write down the advantages and disadvantages of batch operating system.

Ans. **Advantages of Batch Operating System –**

(i) It is very difficult to guess or know the time required by any job to complete. Processors of the batch systems knows how long the job would be when it is in queue.

(ii) Multiple users can share the batch systems.

(iii) The idle time batch system is very less.

(iv) It is easy to manage large work repeatedly in batch systems.

Disadvantages of Batch Operating System –

(i) The computer operators should be well known with batch systems.

(ii) Batch systems are hard to debug.

(iii) It is sometime costly.

(iv) The other jobs will have to wait for an unknown time if any job fails.

Q.20. Why batch systems are used ?

Ans. As batch systems load less stress on processor and involve less user interaction so that is why we can use batch system in current days also. Another advantage of batch systems is that the large repeated jobs are given to the system and we do not have to interact with computer to tell the system that you have to do that job after finishing that job. Old batch systems were not interactive i.e., the user interaction was not involved when the job is running. Now in modern batch systems we have interactions also. For example we can set the timer on the job and when the specific time comes then computer send message to the processor that time is over. This helps us to avoid too many errors and makes us easy to debug.

The batch systems are used by large organizations and also large jobs are done in sequence by the system. So it is best practice to divide the big job into small parts and run them so it is easy to debug the job when error comes. We can also set timer on computer for each job so we can check and interact with job to check and debug errors.

Q.21. Explain briefly the CTSS operating system.

Ans. The term CTSS stands for compatible time-sharing system. It was the first time-sharing operating systems developed at MIT by a group known as project MAC (Machine-Aided Cognition, or Multiple-Access Computers). This system was first developed for the IBM 709 in 1961 and later transferred to an IBM 7094.

The system ran on a computer with 32,000 36-bit words of main memory where as the monitor consumed 5000 of that. The user's program and data were loaded into the remaining 27,000 words of main memory if the control was assigned to an interactive user. The system clock generates the interrupts at a rate of approximately one in every 0.2 seconds. The operating system again gained control and should allocate the processor to another user. This technique is known as ***time slicing***.

For the minimization of disk traffic, the user memory was only written out when the incoming program would overwrite it. This principle is illustrated in fig. 1.7. For example, consider there are four interactive users with the following memory requirements, in words Job1 – 15000, Job 2 – 20000, Job 3 – 5000 and Job 4 – 10000.

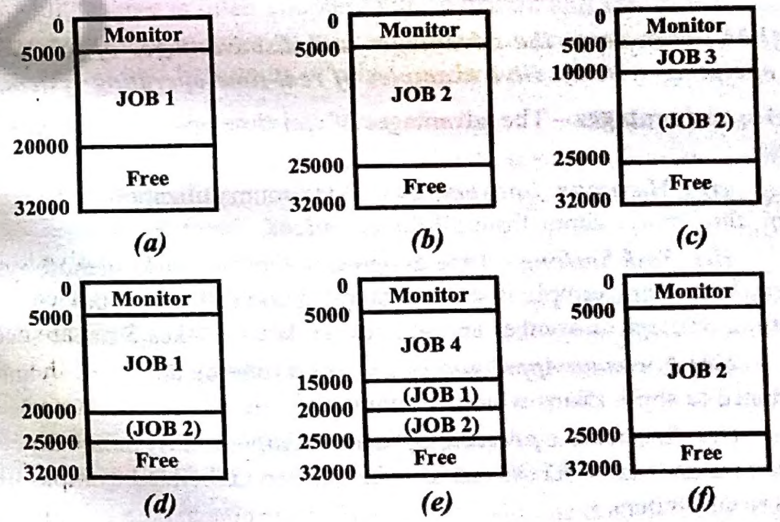


Fig. 1.7 CTSS Operation

The monitor loads Job1 initially and transfers control to it as shown in fig. 1.7 (a). Later the monitor decides to transfer the control to Job 2. Because Job 2 requires more memory than Job 1. Firstly the Job1 must be written out and then Job 2 can be loaded as shown in fig. 1.7(b). Next Job 3 is loaded in to be run. Since Job 3 is smaller than Job 2, a portion of Job 2 can remain in memory reducing disk write time as shown in fig. 1.7 (c). Later, the monitor decides to transfer control back to Job 1. If Job 1 is loaded back into memory, an additional

portion of Job 2 must be written out as shown in fig. 1.7 (d). As Job 4 is loaded, the part of Job 1 and the portion of Job 2 remaining in memory are retained as shown in fig 1.7 (e). At this point if either Job 1 or Job2 is activated, only a partial load will be required. In the given example the Job 2 is running next. It requires that Job 4 and the remaining resident portion of Job1 be written out and that the missing portion of Job 2 be read in as shown in fig. 1.7 (f).

Q.22. Write down the advantages and disadvantages of time-sharing operating systems.

Ans. **Advantages of Time-sharing Operating System –**

(i) Each task gets an equal opportunity.

(ii) Less chances of duplication of software.

(iii) CPU idle time can be reduced.

Disadvantages of Time-sharing Operating System –

(i) Reliability problem.

(ii) One must have to take care of security and integrity of user programs and data.

(iii) Data communication problem.

Q.23. Write down the advantages and disadvantages of RTOS (real-time operating system). Give examples of real-time operating system.

Ans. **Advantages** – The advantages of real-time operating systems are as follows –

(i) Maximum Consumption – Maximum utilization of devices and system, thus more output from all the resources.

(ii) Task Shifting – Time assigned for shifting tasks in these systems are very less. For example in older systems it takes about 10 micro seconds in shifting one task to another and in latest systems it takes 3 micro seconds.

(iii) Focus on Application – Focus on running applications and less importance to applications which are in queue.

(iv) Real-time Operating System in Embedded System – Since size of programs are small, RTOS can also be used in embedded systems like in transport and others.

(v) Error Free – These types of systems are error free.

(vi) Memory Allocation – Memory allocation is best managed in these type of systems.

Disadvantages – The disadvantages of real-time operating systems are as follows –

(i) Limited Tasks – Very few task run at the same time and their concentration is very less on few applications to avoid errors.

(ii) Use Heavy System Resources – Sometimes the system resources are not so good and they are expensive as well.

(iii) Complex Algorithms – The algorithms are very complex and difficult for the designer to write on.

(iv) Device Driver and Interrupt Signals – It needs specific device drivers and interrupt signals to response earliest to interrupts.

(v) Thread Priority – It is not good to set thread priority as these systems are very less pron to switching tasks.

Examples of Real-time Operating Systems – Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

Q.24. Explain the term multitasking. ***(R.G.P.V., Dec. 2008)***

Ans. A multitasking operating system is distinguished by its ability to support concurrent execution of two or more active processes. Multitasking is usually implemented by maintaining code and data of several processes in memory simultaneously and by multiplexing processor and I/O devices among them. Multitasking is often coupled with hardware and software support for memory protection in order to prevent erroneous processes from corrupting address spaces and behaviour of other resident processes.

Q.25. Explain the term multiprograming. ***(R.G.P.V., Dec. 2008)***

Ans. The most important aspect of job scheduling is the ability to multiprogram. In general, a single user cannot keep either the CPU or the I/O devices busy at all times. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The idea is to keep several jobs in memory simultaneously as shown in fig. 1.8. This set of jobs is a subset of the jobs kept in the job pool – since the number of jobs that can be kept simultaneously in memory is usually much smaller than the number of jobs that can be in the job pool. The operating system picks and begins to execute one of the jobs in the memory. Eventually, the job may have to wait for some task, such as an I/O operation to complete. In a non-multiprogrammed system, the CPU would sit idle. In a multiprogramming system, the operating system simply switches to, and executes, another job. When that jobs needs to wait, the CPU is switched to another jobs, and so on. Eventually, the first job finishes waiting and gets the CPU back. The CPU is never idle, as long as at least one job needs to execute.

Operating System
Job 1
Job 2
Job 3
Job 4

Fig. 1.8 Memory Layout for a Multiprogramming System

Q.26. Explain multitasking and multiprogramming. ***(R.G.P.V., Dec. 2012)***

Ans. Refer Q.24 and Q.25.

Q.27. Discuss multiprogramming versus single user systems in terms of throughput and CPU utilization. ***(R.G.P.V., Dec. 2013)***

Ans. In multiprogramming, the physical memory was divided into various partitions, each containing an independent program. One of these partitions was holding the operating system. Since, there was only one CPU, at a time only one program could be executed. Thus, there was a need for a method to switch the CPU from one program to the next. This is what the operating system offered.

One of the major benefits of this mechanism was the increase in the throughput. If various programs were to run one after the other, the total elapsed time would have been much more than under a mechanism which utilized multiprogramming. The reason was simple. In a uniprogramming system, the CPU was idle when any I/O operation for any program was going on, but in a multiprogramming operating system, when the I/O for one program was going on, the CPU was switched to another program. This enabled for the overlapped operations of I/O for one program and the other processing for some other program by the CPU, thereby increasing the throughput.

Q.28. Explain in detail about mainframe systems.

Or

Compare and contrast multiprogramming, batch and time-sharing system. ***(R.G.P.V., Dec. 2013)***

Ans. Mainframe computer systems were the first computers used to tackle many commercial and scientific applications. Now, we trace the growth of mainframe systems from simple batch systems to time-shared systems.

(i) Batch Systems – Early computers were physically enormous machines run from a console. The common input devices were card readers and tape drives. The common output devices were line printers, tape drives, and card punches. The user did not interact directly with the computer systems. The user prepared a job, consisted of the program, the data, and some control information about the nature of the job, and submitted it to the computer operator. The job was usually in the form of punch cards. At some later time, the output appeared. The output consisted of the result of the program, as well as a dump of the final memory and register contents for debugging.

The operating system in these early computers was fairly simple. Its main function was to transfer control automatically from one job to the next. The operating system was always resident in memory as shown in fig. 1.9.

Operating System
User Program Area

Fig. 1.9 Memory Layout for a Simple Batch System

To speed up processing, operators batched together jobs with similar needs and ran them through the computer as a group. Thus, the programmers would leave their programs with the operator. The operator would sort programs into batches with similar requirements and, as the computer became available would run each batch. The output from each job would be sent back to the appropriate programmer.

In this execution environment, the CPU is often idle, because the speeds of the mechanical I/O devices are much slower than electronic devices. Over time, improvements in technology and the introduction of disks resulted in faster I/O devices. However, CPU speeds increased to a even greater extent, thus the problem was not only resolved, but exacerbated.

(ii) Multiprogrammed Systems – Refer Q.25.

(iii) Time-sharing Systems – Refer Q.14.

Q.29. Define a distributed system. ***(R.G.P.V., Dec. 2014)***

Ans. In distributed system or loosely coupled system processors do not share memory, and each processor has its own local memory. Therefore, a processor cannot directly access the memory attached to other processors. Loosely coupled systems use only message passing for interprocessor communication and synchronization. In loosely coupled systems, the processors of distributed computing system can be located far from each other to cover a wider geographical area. In these systems, the intermachine message delay in large and the data rate is low. An example of loosely coupled system is Intel's Hypercube.

Q.30. What is distributed system ? Discuss the advantages of distributed systems. ***(R.G.P.V., Dec. 2016)***

Ans. **Distributed System** – Refer Q.29.

Advantages of Distributed System – Some advantages of distributed system are as follows –

(i) Resource sharing allow many users to share expensive peripherals like color printers.

(ii) A distributed system may have more total computing power than a mainframe.

(iii) These systems are reliable if one machine crashes the system as a whole can still survive.

(iv) It makes human to human communication more easier, for example, by emails.

(v) Data sharing allow many users access to a common database.

Q.31. What is meant by operating systems ? Explain various types of operating systems in detail. ***(R.G.P.V., Dec. 2012)***

Ans. **Operating System** – Refer Q.1.

Types of Operating Systems – Refer Q.9, Q.10, Q.11, Q.12, Q.13, Q.24, Q.25, Q.28, Q.29 and Q.27.

Q.32. Describe the evolution of an operating system from simple batch processing to today's operating system with their advantages and disadvantages. ***(R.G.P.V., Dec. 2010)***

Or

How many types of operating system are there ? Explain each of them. ***(R.G.P.V., June 2010, Dec. 2015)***

Ans. Refer Q.9, Q.10, Q.11, Q.12, Q.13, Q.24, Q.25, Q.28, Q.29 and Q.27.

Q.33. Write down the disadvantages of distributed operating system.

Ans. Following are the disadvantages of distributed operating system –

(i) Failure of the main network will stop the entire communication.

(ii) To establish distributed systems the language which are used are not well defined yet.

(iii) These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet.

Q.34. Write some desirable characteristics and features of operating system.

Ans. Interoperability and ***transparency*** are very desirable characteristics in a computer system. These are so unique characteristics which are used to define network and distributed operating systems. These characteristics provide flexibility and uniformity for the users of the operating system. To balance these two conflicting interests, a solution is to give users convenient option to simulate a secure a private network over the public network without losing too much interoperability and transparency.

Some of the desirable characteristics for real-time operating system are –

(i) Timeliness
(ii) Design for survival under peak load
(iii) Predictability
(iv) Fault-tolerance
(v) Maintainability.

Features of an operating system are –

(i) Hardware interdependency.
(ii) User interface is provided.
(iii) Adaptability of a hardware.
(iv) Management of memory unit.
(v) Managing the task.
(vi) Capability of better work.
(vii) Security to logical access.
(viii) File management.

Q.35. Explain the migration feature of operating system in developing the multiplexed operating system.

Ans. Operating systems examination shows that the features available on mainframes operating system have been adopted by microcomputers. The same concept is applied on other computers such as mainframes, minicomputers, microcomputers and handhelds. The fig. 1.10 shows the concepts of modern operating system, it realize the theme of feature migration and to recognize the long history of many operating system features.

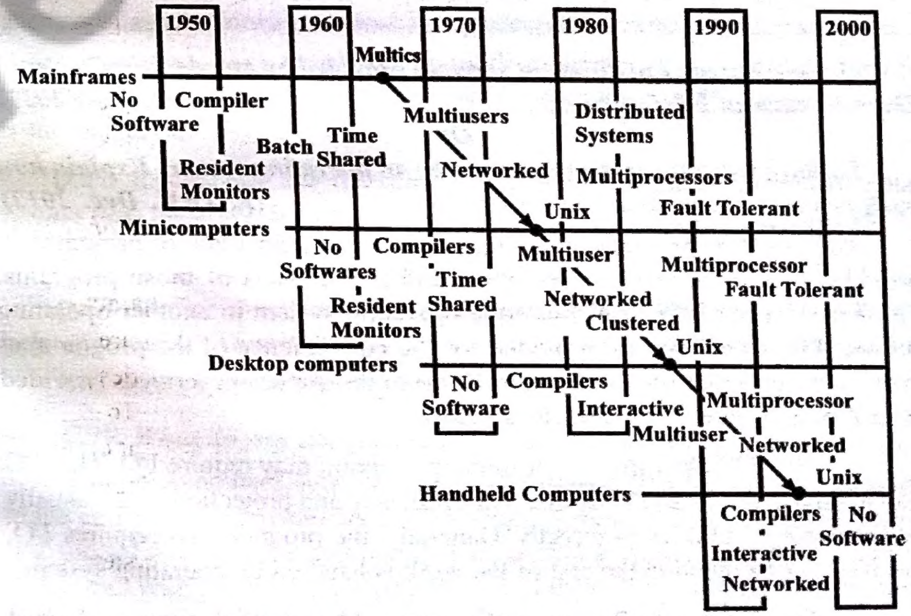


Fig. 1.10

One of the example of feature migration is the multiplexed information and computing services (MULTICS) operating system. It was developed in 1965 to 1970 at the Massachusetts Institute of Technology (MIT) as a

computing utility. It is used to run on the large, complex mainframe computer. Various concepts, subsequently employed at Bell laboratories in the design of Unix, which is developed for MULTICS. The features of UNIX became the basis for UNIX- like operating systems on microcomputer systems and are included in more recent operating systems such as Microsoft Windows NT, IBM OS/2, and the Macintosh operating system. Thus, the features developed for mainframe system have moved to microcomputers.

Therefore the features of large operating systems were being scaled to fit PCs, more powerful, faster and more sophisticated hardware systems were being developed. Some of the examples of large PC are Sun SPARCstation, HP/Apollo, IBM RS/6000, and the Intel Pentium class system running Windows NT or a UNIX derivative.

Many universities and businesses were working on the large number of workstations tied together with local area network.

OPERATING SYSTEMS SERVICES – TYPES OF SERVICES, DIFFERENT WAYS OF PROVIDING THESE SERVICES – UTILITY PROGRAMS, SYSTEM CALLS

Q.36. What are the different services provided by an operating system ? Explain each in brief. ***(R.G.P.V., Dec. 2011)***

Or

Explain various services provided by an operating system. Explain how each provides convenience to the users. ***(R.G.P.V., Dec. 2012)***

Ans. An O.S. provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. These services are different from one operating system to another operating system. These services are provided for the convenience of the programmer to make the programming task easier. Some of the important services provided by operating systems are discussed below –

(i) I/O Operations – A running program may require I/O. This I/O may involve a file or an I/O device. For efficiency and protection, users usually cannot control I/O devices directly. Generally, the program that requires I/O, specifies only a function the rest of the work is handled by operating system.

(ii) Works as a Resource Allocator – When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Various types of resources are managed by operating system. Some may have special allocation code, whereas others may have much more general request and release code.

(iii) File-system Manipulation – Programs need to read and write files. Programs also need to create and delete files by name.

(iv) Error Detection – The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware, in I/O device and in the user program. It is the duty of operating system that display appropriate message whenever an error occurs and takes the suitable action for ensuring correct and consistent computing.

(v) Communications – This is another service provided by operating system. If more than one programs are running on same computer, then there are many instances, when one process needs to exchange information with another process. Such communication can occur in two ways. The first one takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network. Communication may be implemented via shared memory, or by the technique of message passing.

(vi) Protection – Protection involves ensuring that all access to system resources is controlled. When several disjointed processes execute concurrently, it should not possible for one process to interfere with the others, or with the operating system itself. Security of the system from the outsiders is also important. Such security starts with each user having to authenticate himself to the system, usually by means of a password, to be allowed access to the resources.

Q.37. Enumerate the operating system services and functions. Also discuss how these service are provided. ***(R.G.P.V., Dec. 2014)***

Ans. Refer Q.36 and Q.5.

Q.38. Define the operating system architecture and services with the help of suitable example. ***(R.G.P.V., June 2016)***

Ans. Refer Q.1, Q.3 and Q.36.

Q.39. What do you understand by utility program in operating system ?

Ans. In computers, a utility is a small program that provides an addition to the capabilities provided by the operating system. In some usages, a utility is a special and nonessential part of the operating system. The print "utility" that comes with the operating system is an example. It's not absolutely required to run programs and, if it did not come with the operating system, you could perhaps add it. In other usages, a utility is an application that is very specialized and relatively limited in capability. A good example is a search and replace utility. Some operating systems provide a limited capability to do a search-and-replace for given character strings. You can add a much more capable

search-and-replace utility that runs as an application program. However, compared to a word processor, a search-and-replace utility has limited capability.

Q.40. Explain the role of utility programs in operating system.

Ans. All the resources of computer system are manage, maintain and controlled by the utility software. Some of the necessary tools are contained by the operating system for this, but separate utility programs can provide the improved functionality. The utility software is technical and targeted at users with a solid knowledge of computers. Mostly we are using computer for e-mail, sometimes for browsing internet and typing a report, such times we need not have much needs for these utilities. However, if we use the computer for more time, these utilities can help to stay computer in good condition.

Some examples of utility programs are antivirus software, backup software and disk tools.

The name antivirus software suggests the helps to protect a computer system from viruses and from other harmful programs. A computer virus is known as a computer program that can cause damage to a computer's software, hardware or data. It is called as virus because it has the capability to replicate itself and hide inside other computer files.

The most common ways to get a virus is to download the file from the internet. Whereas antivirus software plays an important role, it keeps scanning online activity and checked that the system do not download the infected files. On internet, new viruses are coming out at all the time, therefore it is necessary to update antivirus software very frequently.

Q.41. Why utility programs are so important ? Explain the types of utility program.

Ans. Utility program is a type of system software which creates a workable environment for user to work with application software. Utility program does so by controlling and maintaining the operations of the computer, its devices or its software.

There are many types of utility software a computer can use. Some most important of those are as follows –

(i) Antivirus Software – The software which is used to detect and remove the malwares from computer system is known as antivirus software.

(ii) Disk Defragmenter – The process which reduces the fragmented space in the disk is called the defragmentation. Disk defragmenter program accomplishes this task.

(iii) Disk Cleaner – These are the programs in-built in operating system to find and delete unwanted files from the computer to free the disk space. It could be temporary files or folders automatically created during internet surfing session on the computer.

(iv) Compiler – For making the program executable, a programming language uses a program to transform the source code written in programming language into machine language code is known as compiler.

Q.42. Write the differences between application software and utility programs.

Ans. The differences between application software and utility programs are as follows –

(i) An application software is a computer based program that is designed to perform some tasks that are grouped together and helps people in completing their work at faster speeds. Whereas a utility program is a program that is designed to perform specific tasks that contribute to making the device work better and keeping the environment safe.

(ii) An application software is mostly downloaded from the internet whereas a utility program is either already installed in the computer of becomes downloadable from the web.

(iii) The best example of an application software is that of a video player that is used to play various files on the computer. While the best example of a utility program is the antivirus program installed on the computer that keeps the computer safe.

(iv) An application usually has bigger size takes more space on the computer whereas a utility is smaller in size and takes less space and power in comparison.

(v) An application becomes an accessory that may or may not show any benefit whereas a utility always shows some benefit towards the device.

(vi) A utility is always free if installed in the computer, whereas an application mostly paid if downloaded from the web.

Q.43. Write short note on system calls.

(R.G.P.V., Dec. 2004, June 2010, May 2018)

Ans. The interface between the operating system and the user programs in defined by the set of "extended instructions" that the operating system provides. These extended instructions are known as ***system calls***.

User programs communicate with the operating system and request services from it by making system calls. There exists a library procedure corresponding to each system call that user programs can call. This procedure puts the parameters of the system call in a specified place, such as the machine registers, and then issues a TRAP instruction to start the operating system. When the operating system gets control after the TRAP, it examines the parameters to see if they are valid, and if so, performs the work requested. When it is finished, the operating system puts a status code in a register, telling whether it succeeded of

failed, and executes a RETURN FROM TRAP instruction, to return control back to the library procedure. The library procedure then returns to the caller in the usual way, returning the status code as a function value.

Q.44. What is the purpose of system calls ? ***(R.G.P.V., Dec. 2015)***

Ans. System calls allow user-level processes to request services of the operating system.

Q.45. What is meant by a system call ? How it can be used ? How does an application program use these calls during execution ? How is all this related to the compilation process ? ***(R.G.P.V., Dec. 2013)***

Or

What do you understand by system call ? Explain its uses with the help of example. ***(R.G.P.V., June 2016)***

Ans. **System Call** – Refer Q.43.

System calls are callable from the assembly language. We have often encountered phrase such as "assembly language under DOS". We should now be clear about its meaning. The assembly language programmer needs to know the assembly language instructions set for that machine. In addition, he normally needs to be aware of the system calls of the operating system running on that machine. This is the real meaning of this phrase. Many high level languages such as C also enable system calls to be embedded explicitly in the midst of other statements. Some high level languages such as Java or C++ do not need the explicit use of system calls embedded in other statements. In these scenarios, the compiler substitutes system calls at specific locations, wherever necessary, for the suitable instructions in high level language.

As you go higher in the level of languages such as 3GLs or 4GLs, the less will you require to explicitly use system calls. However, the compilers of these languages almost definitely will use different system calls in the compiled programs. The two compilers will be substituting various sets of operating system calls for the same source statements such as fread, fwrite or exit. Apart from the format, the functionality, parameters passed, returned results, scope and errors differ quite a lot between the two operating systems. Generally, the mapping between the system calls of two operating systems is never straightforward. Normally, before the real system call, the compiler produces machine instructions which are preparatory instructions for that system call. Typically, these instructions load certain CPU registers or the stack with the parameters of the system call.

The system call, while executing, considers that the parameters will be present at a predefined location. The results of the system call or the error

codes are stored by the system call also in predefined registers or the stack. The instructions following the system call produced by the compiler pick up these results from these registers and then proceed.

Q.46. Describe system call and its types. ***(R.G.P.V., Dec. 2010)***

Or

What are system calls ? Explain briefly about various types of system calls provided by an operating system. ***(R.G.P.V., Dec. 2014)***

Ans. **System Call** – Refer Q.43.

Types of System Calls – System calls can be grouped into five categories – process control, file management, device management, information management and communications.

(i) Process Control – A process or job executing one program may want to ***load*** and ***execute*** another program. This feature allows the command interpreter to execute a program as directed by, for example, a user command, the click of a mouse. An interesting question arises that where to return control when the loaded program terminates. If control returns to the existing program when the new program terminates, the memory image of the existing program must be saved. Thus, we have effectively created a mechanism for one program to call another program. If both programs continues concurrently, we have created a new job or process to be multiprogrammed. The system calls for this purpose are ***create process*** or ***submit job***.

If a new job or process, or a set of jobs or processes are created, we should be able to control its execution. This control requires the ability to determine and reset the attributes of a job or process, including the job's priority, its maximum allowable execution time, and so on ***(get process attributes*** and ***set process attributes)***. It is also needed to terminate a job or process that we created ***(terminate process)*** if it is incorrect or is no longer needed.

After creating new jobs or processes, it is needed to wait for them to finish their execution. We may want to wait for a certain amount of time ***(wait time)***. More likely, we may want to wait for a specific event to occur ***(wait event)***. The jobs or processes then signal when that event has occurred ***(signal event)***.

(ii) File Management – We need to able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes. Once the file is created, we need to open it and to use it. We may also read, write, or reposition. Finally, we need to close the file, indicating that it is no longer used.

We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system. For files or directories, we need to be able to determine the values of various attributes and to reset

them. File attributes are the file name, a file type, protection codes, accounting information and so on. Two system calls, ***get file attribute*** and ***set file attribute***, are required for this function.

(iii) Device Management – A running program may need additional resources to proceed such as memory, access to files and so on. If the resources are available, they are granted, and control is returned to the user program. Otherwise the program will have to wait until resources are free.

Files are thought of as abstract or virtual devices. Thus, many of the system calls for files are also needed for devices. If the system has multiple users, we must first ***request*** the device to ensure exclusive use of it. After we are finished with the device, we must ***release*** it. Once, the device has been requested and allocated to us, we can read, write, and reposition the device.

(iv) Information Management – Many system calls exist for the purpose of transferring information between the user program and the operating system. For example, most systems have a system call to return the current ***time*** and ***date***. Other system calls may return information about the system, such as the number of current users, the version number of operating system, the amount of free memory or disk space, and so on.

(v) Communication – There are two common models of communication. In the ***message-passing model***, information is exchanged through an interprocess-communication facility provided by the operating system. A connection is opened before communication to take place. The name of the other communicator must be known, whether it is another process on the same CPU, or a process on another computer in communications network. Each computer in a network has a ***host name***. Similarly, each process has a ***process name***, which is translated into an equivalent identifier for referring to it by the operating system. The ***get hostid*** and ***get processid*** system calls do this translation. These identifiers are then passed to the general purpose open and close calls provided by the file system, or to specific open connection and close connection system calls. The recipient process gives its permission for communication to take place with an ***accept*** connection call. Most processes that will be receiving connections are special purpose daemons. They execute a ***wait for connection*** call and are awakened when a connection is made. The source of the communication, known as the client, and the receiving daemon, known as a server, then exchange messages by read message and write message system calls. The close connection call terminates the communication.

In the ***shared-memory model***, processes use map memory system calls to gain access to regions of memory owned by other processes.

Q.47. What is system booting ? Explain.

Or

Define boot block. ***(R.G.P.V., Dec. 2012)***

Or

Explain various steps involved in booting. ***(R.G.P.V., Dec. 2013)***

Or

Explain the booting process. ***(R.G.P.V., Dec. 2014)***

Ans. Whenever a computer system is "cold started", say, after being powered or following a system crash, at least a portion of the operating system must be brought into main memory and given control of the processor. This activity is known as ***system booting*** or ***bootstrapping*** of an operating system. Typically, the hardware initially transfers control to a known address where a starting routine in ROM is placed. This routine is called the ***bootstrap loader.*** It can be used to bring the rest of the system gradually to main memory, for instance, from secondary memory or from another node in a distributed system. In disk-based systems, the core portion of the operating system is often placed at a known address, called the ***boot block*** or ***boot area*** of a known system disk drive. Thus, the bootstrap loader routine can include a rudimentary form of a disk driver whose primary function is to load and to activate the initialization section of the operating system. This section can, in turn, load into main memory the rest of the operating system and complete the initialization process. In memory-based systems, such as KMOS, the starting routine can simply transfer control to the operating system, which itself may be residing in ROM.

Q.48. Compare spooling and buffering. ***(R.G.P.V., Dec. 2011, 2012)***

Or

Explain spooling and buffering. ***(R.G.P.V., Dec. 2012)***

Or

What is buffering and spooling ? ***(R.G.P.V., June 2016)***

Ans. **Spooling** – A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. ***Spooling*** is the process in which jobs from the cards are read directly to the disk and the location of that card in the disk is recorded in a table by the operating system. When that job is required for execution, it is read from the disk (see fig. 1.11).

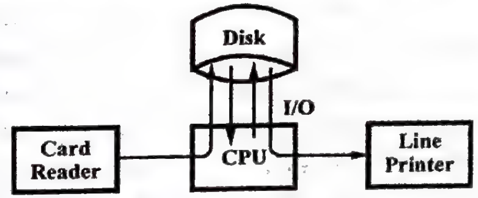


Fig. 1.11 Spooling

Similarly, when job requires a printer for output a line, that particular line is copied onto the disk and system buffer, and at the completion of job output is read from disk and printed by the printer.

By spooling more than one input and output devices can interact with disk simultaneously and provide an impression that these devices interact directly with CPU, so it is ***called simultaneous peripheral operation on line***.

Some devices, such as tape drive and printers, cannot usefully multiplex the I/O requests of multiple concurrent applications. Spooling is one way that OS can coordinate concurrent output. Another way to deal with concurrent device access is for providing explicit facilities for coordination. Some OS provide support for exclusive device access, by enabling a process to allocate an idle device, and deallocated that device when it is no longer needed.

Buffering – A buffer is a memory area that stores data while they are transferred between two devices or between a device and an application. Buffering is done for three reasons. First reason is to cope with a speed mismatch between consumer and producer of a data stream. Suppose, for example, that a file is being received via modem for storage on the hard disk. Thus, a buffer is created in main memory to accumulate the bytes received from modem. When an entire buffer of data has arrived, the buffer can be written to disk in a single operation.

However the disk write is not instantaneous and the modem still needs a place to store additional incoming data, two buffers are used. When the first buffer will fill, then the disk write is requested. Now modem starts to fill the second buffer, while the first buffer is written to disk. After finishing the disk write from first buffer, modem can switch back to first buffer while disk write from second one. This is known as ***double buffering.*** This double buffering decouples the producer of data from the consumer, thus relaxing timing requirements between them.

The second use of buffering is to adapt between devices that have different data-transfer sizes. Such disparities are especially common in computer networking, where buffers are used widely for fragmentation and reassembly of messages. A large message is fragmented into small network packets, at sending side. These packets are sent over the network, and the receiving side places them in a reassembly buffer to form an image of the source data.

Third reason of buffering is to support copy semantics for application I/O. Now consider an application has a buffer of data that it wishes to write

on disk. It calls the "write ( )" system call providing a pointer to the buffer and an integer specifying the number of bytes to write. With ***copy semantics***, the version of the data written to disk is guaranteed to be the version at the time of the application system call, independent of any subsequent changes in the application's buffer. The disk write is performed from the kernel buffer, so that subsequent changes to the application buffer have no effect. Copying of data between kernel buffers and application data space is common in operating systems, despite the overhead that this operation introduces because of the clean semantics. The same effect can be obtained more efficiently by clever use of virtual-memory mapping and copy-on-write page protection.

Q.49. What is spooling ? What are the advantages of spooling over buffering ? ***(R.G.P.V., June 2011)***

Ans. **Spooling** – Refer Q.48.

The only advantage of spooling was that you no longer had to carry tapes to and fro the 1401 and 1709 machines. Under the new operating system, all jobs in the form of cards could be read into the disk first and later on, the OS would load as many jobs in the memory, one after the other, until the available memory could accommodate them. After many programs were loaded in different portions of the memory, the CPU was switched from one program to another to achieve multiprogramming. Spooling allowed smooth multiprogramming operations.

The I/O of all the jobs was essentially pooled together in the spooling method and therefore this could be overlapped with the CPU bound computations of all the jobs at an appropriate time chosen by the OS to improve the throughput.

Q.50. Discuss how CPU protection is provided by an operating system.

Ans. We must prevent a user program from getting stuck in an infinite loop or not calling system services, and never returning control to the operating system. This goal is achieved by using a ***timer.*** A timer can be set to interrupt the computer after a specified period. The period may be fixed or variable. A variable timer is generally implemented by a fixed-rate clock and a counter. The operating system sets the counter. Every time the clock ticks, the counter is decremented. When the counter reaches 0, an interrupt occurs. For example, a 10-bit counter with a 1 millisecond clock allows interrupts at intervals from 1 millisecond to 1024 milliseconds, in steps of 1 millisecond.

Before turning over control to the user, the operating system ensures that the timer is set to interrupt. If the timer interrupts, control transfers automatically to the operating system, which may treat the interrupt as a fatal error or may give the program more time. Obviously, instructions that modify the operation of the timer are privileged.

Thus, a timer can be used to prevent a user program from running too long. A simple technique is to initialize a counter with the amount of time that a program is allowed to run. For example, a program with a 6 minute time limit would have its counter initialized to 360. Every second, the timer interrupts and the counter is decremented by 1. As long as the counter is positive, control is returned to the user program. When the counter becomes negative, the operating system terminates the program for exceeding the assigned time limit.

⌘ ⌘

UNIT 2

FILE SYSTEMS – FILE CONCEPT, USER'S AND SYSTEM PROGRAMMER'S VIEW OF FILE SYSTEM, DISK ORGANIZATION, TAPE ORGANIZATION

Q.1. What is a file system ?

Ans. A *file system* is a collection of files. It consists of two words file and system. The word 'file' signifies a bunch of data put somewhere. But the important question is where ? If, for example, in a library the librarian starts keeping the books in a disorganized fashion, it will be very difficult to find a book later on. This is where 'system' comes in – instead of just throwing the data to the device, we generalize and construct a system that virtualizes a nice and ordered structure in which we can arrange our data in much the same way as books are arranged in a library. The purpose of the file system is to make it easy for us to update and maintain our data.

File systems allow users to organize files and other file system objects through the use of directories. A directory (or folder) has been defined to be a file system object that contains other file system objects. However, in most cases, it is more accurate to define a directory as an object that contains the names of file system objects.

Q.2. Discuss the concept of file.

Or

What is file ? ***(R.G.P.V., June 2016)***

Ans. A file is a named collection of information that recorded on secondary storage. From a user's perspective, a file is the smallest allotment of logical secondary storage. It means that data cannot be written to secondary storage unless they are within a file. Commonly, files represent programs and data. Data files may be numeric, alphanumeric, or binary. In general, a file is a sequence of bits, bytes, lines, or records.

The information in a file is defined by its creator. The following different types of information may be stored in a file-source programs, object programs,

executable programs, numeric data, text, payroll records, graphic images, sound recordings and so on.

Q.3. Enlist and describe attributes of files.

Ans. A file has certain attributes, which vary from one operating system to another. The common attributes are as follows –

(i) File Name – This is the name of file that can be read by the user. It means user identifies file by file name attribute.

(ii) Type – If system is supporting more than one type of files, type attribute is required. Different extensions are used to specify the type of file like .obj, .exe, etc.

(iii) Size – This attribute contains size of file in bytes or blocks.

(iv) Location – This is the address of location where file stored in the device.

(v) Protection – This attribute contains information that protect file from unauthorized users. Who will read, write, and execute the file is specified by protection attribute.

(vi) Time and Date – This information tells, when file is created and last modified and last use. This information is very valuable for usage monitoring.

(vii) Lock Flag – Lock flag attribute is used to lock the file from unwanted access. Here, 0 is used for unlock and nonzero for locked.

(viii) Hidden Flag – Hidden flag attribute is used to hide the file. If flag is set to 1, it means that file will not be displayed in listing.

(ix) System Flag – System flags are used to make a system file. One is used to set system flag, it shows that file is a system file.

(x) Maximum Size – This attribute is used to give information about the maximum size of file attainable to user.

(xi) Record Length – It is used to provide informations about the length of record to user.

Q.4. What are the operations possible on the file ? (R.G.P.V., Dec. 2014)

Ans. Following are some basic operations on file –

(i) Creating a file
(ii) Reading from file
(iii) Writing to file
(iv) Repositioning within a file
(v) Deleting a file
(vi) Truncating a file
(vii) Open a file
(viii) Close a file
(ix) Append a file
(x) Seek
(xi) Rename a file
(xii) Set attributes
(xiii) Get attributes.

Q.5. Describe some common file types with their extension and function in brief.

Ans. Table 2.1 gives the brief description of some common file types with their extension and function.

Table 2.1 Common File Types

S.No.	***File Type***	***Usual Extensions***	***Functions***
(i)	Multimedia	mpeg, dat, mov, rm	Binary file containing audio and video data.
(ii)	Executable	exe, com, bin, etc.	Made to run machine language program.
(iii)	Sound	mpeg, sum,wav,au	Binary file containing audio data.
(iv)	Source code	C, CC, Java, pas, asm, html, wml	It contains source code in various languages.
(v)	Text	txt, doc, ws, etc.	It contains textual data and documents.
(vi)	Library	lib, a, so, au, mpeg, mov, rm, etc.	It contains libraries of routines for programmers.
(vii)	Print or view	arc, zip, tar	ASCII or binary files in a format for printing or viewing. These are compressed files.
(viii)	Archieve or Compressed	arc, zip, tar	Selected files grouped into one file, compressor sometimes compressed for archieving or storage.
(ix)	Batch	bat, sh	It contains set of commands to command interpreter.
(x)	Object	obj, o, class, etc.	Compiled, machine language, not linked.
(xi)	Word processor	wp, pdf, tex, rrt doc, etc.	It contains various word-processor formats.
(xii)	Image	jpeg, bmp, gif, pcx etc.	It contains information of image in the form of binary.

Q.6. Explain the user's and system programmer's view of file system.

***Ans.* User View** – The system interface which is employed by the users is responsible for user view. The different types of user view are as follows –

(i) The operating system is largely designed to make the interaction easy when user uses a personal computer. There is no need for the operating system to worry about resource utilization.

(ii) The operating system is largely concerned with resource utilization, if the user is using a system connected to mainframe or a minicomputer. It is because there multiple terminals connected to the mainframe and the operating system makes sure that all the resources such as CPU, memory, I/O devices etc. are divided uniformly between them.

(iii) If the user is operating on a workstation connected to other workstations through networks, then the operating system needs to focus on both individual uses of resources and sharing though the network. It happens because the resources used by the one workstation should share the files with other workstation across the network.

(iv) If the user uses mobile phones, then the operating system handles the device usability, including a few remote operations. The battery level of the device is also taken into account.

System View – The operating system is the bridge between applications and hardware according to the computer system. It is compatible with the hardware and is used to control it as needed.

The different types of system view for operating system are as follows –

(i) The operating system is viewed by the system as a resource allocator. The resources are CPU time, memory space, file storage space, I/O devices etc. that are required by processes for execution. It is the duty of the operating system to allocate these resources to the processes judiciously so that the computer system can run fast and smooth.

(ii) The operating system work as a control program. All the processes and the I/O devices are managed by the operating system so that computer system can work smoothly without any error. It makes sure that the I/O devices work in a proper manner without creating problems.

(iii) Operating systems are viewed as a way to make using hardware easier.

(iv) Computers are needed to solve the user problems easily. However, to work with computer hardware directly, it is not an easy task. Therefore operating systems were developed to communicate with the hardware easily.

(v) An operating system is a program running in the background of a computer system all the times and handling all the application programs.

Q.7. Explain in detail about the concept of disk organization.

Ans. A hard disk is a memory storage device which looks like as shown in fig. 2.1.

The disk is divided into tracks. Each track is further divided into sectors. The point to be noted here is that outer tracks are bigger in size than the inner tracks but they contain the same number of sectors and have equal storage

capacity. This is because the storage density is high in sectors of the inner tracks where as the bits are sparsely arranged in sectors of the outer tracks. Some space of every sector is used for formatting. So, the actual capacity of a sector is less than the given capacity.

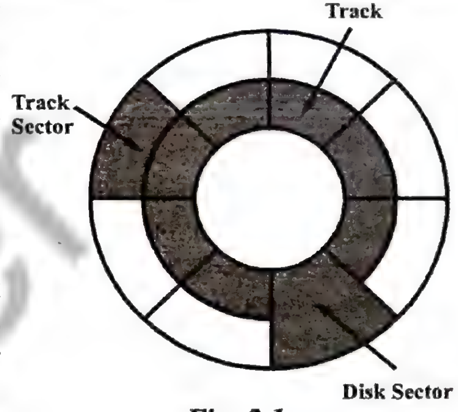


Fig. 2.1

Read-Write (R-W) head moves over the rotating hard disk. It is this Read-Write head that performs all the read and write operations on the disk and hence, position of the R-W head is a major concern. To perform a read or write operation on a memory location, we need to place the R-W head over that position. Some important terms must be noted here –

(i) Seek Time – The time taken by the R-W head to reach the desired track from it's current position.

(ii) Rotational Latency – Time taken by the sector to come under the R-W head.

(iii) Data Transfer Time – Time taken to transfer the required amount of data. It depends upon the rotational speed.

(iv) Controller Time – The processing time taken by the controller.

(v) Average Access Time – Seek time + Average rotational latency + Data transfer time + controller time.

Q.8. Explain in detail about disk management system.

Ans. The operating system manage particular part of disk management. Like, Disk Formatting, booting and bad-block recovery.

Disk Formating – A new magnetic disk is a blank slate. It is partitioned into sectors that the disk controller can read and write before storing data. This process is known as physical/low-level formatting. For each sector low-level formatting fills the disk with a special data structure. This data structure consists of a header, a data area, and a trailer. The header and trailer stores information like sector number and an error correcting code that is used by the disk controller. The error correcting code is changed with a value determined from all the bytes in the data area when the controller writes a sector of data during normal I/O. The error correcting code is recalculated and compared with the stored value while reading the sector. If a mismatch is found, it specifies that the data area of the sector has become corrupted and that the disk sector may be bad. As a part of the manufacturing process, most hard disks are low-level formatted. This formatting enables the manufacturer to test the disk and to initialize the mapping from logical

block numbers to defect free sectors on the disk. The operating system sti needs to record its own data structures on the disk to use a disk to store files. It can be performed in two steps. In first step, the disk is partition into one or more groups of cylinders. The second step is logical formatting in which the operating system stores the initial file-system data structures onto the disk.

Booting – Refer Q.47, Unit-I.

Bad Blocks – The disk have some part are movable and some tolerances. It is the reason to disk failure. Sometimes disk is failure and we need to replace it and we have to need to store data in other disk for recovery and then replace failure disk with new disk and restore back data from other disk in which we stored data. This process is called backup. After taking backup the disk is ready for work. More disks are come from factory with ***bad blocks***. And sometime bad blocks are create by type of using. How is controller using disk. These blocks are handle in many way like if disk is come with IDE controllers, bad blocks are handle manually by controller or user. The MS DOS has command for formate the disk logically called "FORMAT COMMAND". It also scan bad block. If we found bad block it writes a special code into the related FAT entry to tell the allocation routines please do not use that block because it is bad block. If blocks go bad at the time of normal operation, a special program run for scan and lock that block manually. If bad block occurs, data place in bad blocks are loss.

High-end PCS and most work stations and servers are smarter about bad block recovery and it is used SCSI disks. Controller maintains the list of bad block and it is initialize at the time of low level formatting in factory and it update the life of disk. Low level formatting is not visible in OS. The controller can tell to replace each bad sector logically with one of the spare sectors. This is called "forwarding or sector sparing".

Q.9. How the total data access time of a disk is determined ?

(R.G.P.V., Dec. 2006)

Ans. When the disk drive is operating, the disk is rotating at constant speed. To read or write, the head must be positioned at the desired track and at the beginning of the desired sector on that track. Track selection involves moving the head in a movable-head system or electronically selecting one head on a fixed-head system. On a movable-head system, the time it takes to position the head at the track is known as ***seek time***. In either case, once the track is selected, the disk controller waits until the appropriate sector rotates to line up with the head. The time it takes for the beginning of the sector to reach the head is known as ***rotational delay***, or ***rotational latency***. The sum of the seek time, if any, and the rotational delay is the **access time**, the time it takes to get into position to read or write. Once the head is in position, the read or write operation is then performed as the sector moves under the head; this is the data transfer portion of the operation.

Q.10. Define the term disk reliability. ***(R.G.P.V., Dec. 2015)***

Ans. Disk reliability is a more researched area because data stored in magnetic disks tends to be more resistant to transient errors than data stored in solid state memories. In other words, whereas reliability in solid state memories is largely connected with correcting soft errors, reliability in hard disks is concerned with the fact that disks occasionally die, taking most or all of their data with them.

Q.11. Explain the process of disk controller and driver in transferring the data IN or OUT. Define the primary function of disk controller.

Ans. Disks are capable of carrying out only rather primitive commands because of they are electromechanical devices. The interface signals between a disk drive and the disk controller of a computer system are shown in fig. 2.2. Here it is given that the controller is capable of handling several drives with similar characteristics. Some of the control lines are needed to select the drive designated to participate in a given operation. The control lines shown as DRIVE SELECT lines as shown in fig. 2.2. Same as the HEAD SELECT lines are used to activate a specific head on the selected drive. For the moving of head-drives the DIRECTION signals are used to designate the direction, IN or OUT, so that the heads should be moved from the current position. STEP line is used to provide a timed sequence of step pulses. Generally the head of one cylinder is moved on the movement of one pulse, and a predetermined number of pulses moves the head assembly from its present cylinder to the target cylinder.

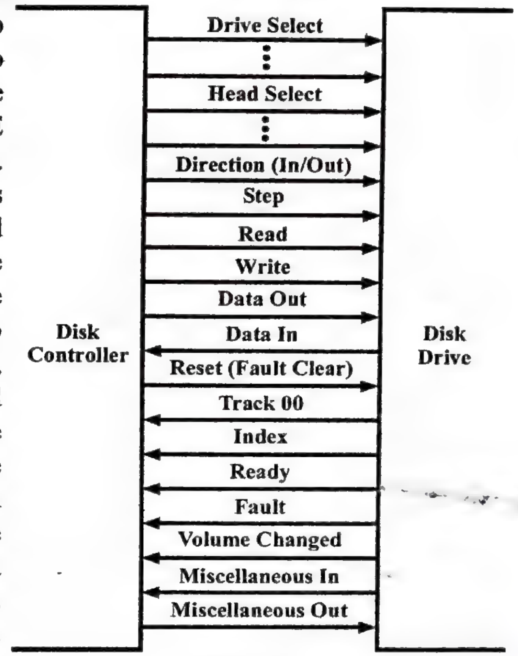


Fig. 2.2

To activate the selected read/write head, the READ and WRITE signals are used. The input or the output stream of bits are carried out by the DATA IN and DATA OUT lines, when a READ or a WRITE operation is in progress, respectively. The drive-supplied signal TRACK 00 which shows that when the head assembly is on the cylinder 0 track 0, the outermost or home position. The time is shown by the INDEX signal, when the drive electronics senses

the cylinder or track address mark (index hole on floppy disks). The DISI CHANGE signal, usually used for removable media, alerts the operating syster to media changes. On the detection of this event, the operating system mu invalidate in main memory all information regarding the related drive, lik directory entries and free-space tables.

Some other signals contains RESET and FAULT indication and a fe other device specific signals collectively labeled as MISCELLANEOUS I and OUT. Some examples are MODE of encoding DOOR OPEN, MOTOI ON, and WRITE PROTECT.

The primary functions of a basic disk controller are as follows –

(i) Disk controller is used to convert higher-level commands, lik SEEK or READ a sector, into a sequence of properly timed drive-specifi commands.

(ii) Disk controller give serial-to-praralle1 conversion and signa conditioning essential to convert from byte or word format, needed for DMA communication with main memory into the analog bit-serial streams expecte and obtain by disk drives.

(iii) It is used to perform error checking and conrol.

Q.12. Explain the tape organization in operating system.

Ans. **File** – A file is a named collection of related information which i recorded on secondary storage like magnetic disks, magnetic tapes and optica disks. Generally, a file is a sequence of bits, bytes, lines or records whos meaning is defined by the files creator and user.

File Structure – A file structure should be according to a required forma that the operating system can understand.

(i) The structure of file is certainly defined according to its type.

(ii) A text file is a sequence of characters organized into lines.

(iii) A source file is a sequence of functions and procedures.

(iv) An object file is considered as a sequence of bytes organized into blocks that are understandable by the machine.

(v) When the different file structures are defined by the operating system, it contains the code to support these file structures. Operating system such as UNIX, MS-DOS support minimum number of file structure.

File Type – File type refers to the ability of the operating system to distinguish different types of file like text files, source files and binary files etc. Different types of operating systems support different types of files. Some operating systems such as MS-DOS and UNIX have the following

files types –

(i) Ordinary Files –

(a) These types of files contain the user information.

(b) These may contain the text, databases or executable programs.

(c) Various operations can be applied on such files like add, modify, delete or even remove the entire file.

(ii) Directory Files – These files show the list of file names and other information related to these files.

(iii) Special Files –

(a) These files are called device files.

(b) The special files are used to represent physical device such as disks, terminals, printers, networks, tape drive etc. These are of two types –

(1) Character Special Files – In this type of file, data is handled character by character as in case of terminals or printers.

(2) Block Special Files – In these types of files, data is handled in blocks as in the case of disks and tapes.

NUMERICAL PROBLEMS

Prob.1. Consider a hard disk with –

4 surfaces

64 tracks/surface

128 sectors/track

256 bytes/sector.

Calculate –

(i) What is the capacity of the hard disk ?

(ii) If the disk is rotating at 3600 r.p.m., then what is the data transfer rate ?

(iii) If the disk is rotating at 3600 r.p.m., what is the average access time ?

[If the seek time and controller time is not mentioned take them to be zero]

Sol. (i) Disk capacity = Surfaces × Tracks/surface × Sectors/track × Bytes/sector

$$= 4 \times 64 \times 128 \times 256 = 8388608$$

Disk capacity = **8 MB Ans.**

(ii) Rotation of disk = 3600 r.p.m.

i.e., 60 sec = 3600 rotations

$\therefore$ 1 sec = 60 rotations

Data transfer rate = Number of rotations per second × Track capacity × Number of surface

[∴ 1 R-W head is used for each surface]

∴ Data transfer rate = 60 × 128 × 256 × 4

Data transfer rate = **7.5 MB/sec** **Ans.**

(iii) Since, seek time, controller time and the amount of data to be transferred is not given, therefore we consider all the three terms as 0.

∴ Average access time = Average rotational delay

Rotational latency = 60 sec = 3600 rotations

1 sec = 60 rotations

Rotational latency = (1/60) sec = 16.67 msec

Average rotational latency = (16.67)/2 = 8.33 msec

Average access time = **8.33 msec** **Ans.**

DIFFERENT MODULES OF A FILE SYSTEM, DISK SPACE ALLOCATION METHODS – CONTIGUOUS, LINKED, INDEXED

Q.13. What is file-system module ? Explain the different modules of a file system.

Ans. The file-system module is part of the system library and provides a generic interface to the file system of the local machine. Remotely mounted file systems are also accessible using this module.

Types Specific to File System Operations – The file-system module contains a number of types specifically designed for use by interfaces in the module, such as –

<file-type>,
<pathname>,
<copy/rename-disposition>.

Manipulating Files – The file-system module contains a number of interfaces that let you perform standard file management operations on files already resident on the file system. You can rename, copy, or delete any file, and you can set any available properties for the file, for example

copy-file,
delete-file,
rename-file,
file-property-setter etc.

Manipulating Directories – The file-system module contains a number of interfaces that let you create and delete directories. These can be used in conjunction with the file manipulation operations described in manipulating

files to perform file management tasks at any position in the file system, for example

create-directory,
delete-directory,
ensure-directories-exist,
do-directory,
working-directory-setter, etc.

Example of file system modules are –

(i) Directory module – Relates file names to file IDs
(ii) File module – Relates file IDs to particular files
(iii) Access control module – Checks permission for operation requested
(iv) File access module – Reads or writes file data or attributes
(v) Block module – Accesses and allocates disk blocks
(vi) Device module – Disk I/O and buffering

Q.14. Explain in detail about various ways of free space management. (R.G.P.V., Dec. 2012)

Or

Discuss various free space management techniques. Compare their advantages and disadvantages. (R.G.P.V., Dec. 2014)

Or

Explain free space management technique. (R.G.P.V., Dec. 2015, June 2016)

Ans. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks. To create a new file, the free-space list is searched for the required amount of space, and this space is allocated to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list. The various approaches to implement free space list are –

(i) Bit Vector – The free-space list is implemented as a bit map or bit vector. Each block is represented by one bit. If the block is free, the bit is 1. If the block is allocated, the bit is 0.

For example, a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, 27 are free, and the rest of blocks are allocated. The free-space bit map will be

001111001111110001100000011100000.....

The main advantage of this approach is its relatively simplicity and efficiency in findings the first free block, or n consecutive free blocks on the disk.

Unfortunately, bit vectors are inefficient unless the entire vector is kept in main memory. Keeping it in main memory is possible for smaller disks, such as on microcomputers, but not for larger ones.

(ii) Linked List – This approach links all the free disk blocks together, keeping a pointer to the first block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free block, and so on.

For example, a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26, and 27 are free, and the rest of the blocks are allocated. A pointer points to block 2, as the first free block. Block 2 in turn contains a pointer to block 3, which will point to block 4, and so on.

However, this scheme is not efficient, to traverse the list, we must read each block, which requires substantial I/O time. Fortunately, traversing the free list is not a frequent action. Usually, the operating systems needs a free block so that it can allocate that block to a file, so the first block in the free list is used.

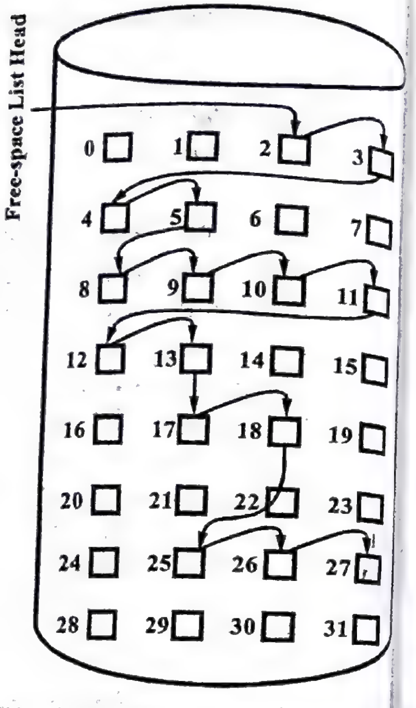


Fig. 2.3 Linked Free-space List on Disk

(iii) Grouping – Grouping approach is the modification to the free-list approach. This method stores the addresses on n free blocks in the first free block. Thus, the first n – 1 of these blocks are free. The last block contains the addresses of another n free blocks, and so on. The importance of this approach is that the addresses of a large number of free blocks can be found quickly, unlike the link-list approach.

(iv) Counting – Counting approach uses the fact that several contiguous blocks may be allocated or freed simultaneously, when space is allocated with the contiguous allocation algorithm or clustering. Thus, rather than keeping a list of n free disk addresses, keep the address of the first free block and the number n of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than a simple disk address, the overall list will be shorter, as long as the count is greater than 1.

Q.15. Explain various file allocation methods in detail.
(R.G.P.V., Dec. 2010)

Or

What are various file allocation methods ? Explain linked allocation method in detail.
(R.G.P.V., Dec. 2012)

Or

Describe various space allocation strategies with their merits/demerits.
(R.G.P.V., Dec. 2013)

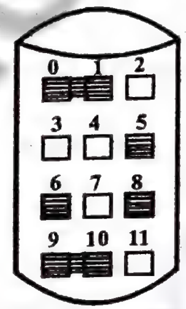
Or

Explain the following file allocation schemes –
(i) Contiguous (ii) Linked (iii) Index-allocation.
(R.G.P.V., Dec. 2008)

Ans. The methods of allocating disk space are as follows –

(i) Contiguous Allocation Method – The simplest allocation scheme is to store each file as a set of contiguous blocks on the disk. Thus, on a disk with 1K blocks, a 50K file will be allocated 50 consecutive blocks.

Contiguous allocation of a file is defined by the disk address and length of the first block. If the file is n blocks long and starts at location b, then it occupies blocks b, b + 1, b + 2, ..., b + n – 1. The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file, as shown in fig. 2.4.



Directory

File	Start	Length
Count	0	2
New	8	3
List	5	2

Fig. 2.4 Contiguous Allocation of Disk Space

Advantages –

(a) It is the simplest allocation scheme and can be implemented easily. Only file size and first starting block number is necessary to remember entire file.

(b) Since entire file can be read in a single operation, so it gives excellent performance.

Disadvantages –

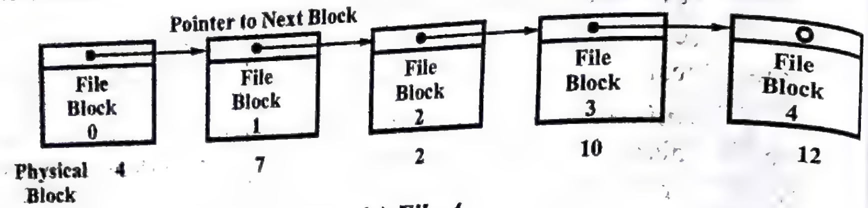
(a) The size of file up to maximum limit must be known at the creation time otherwise this policy not be implemented.

(b) Some disk space is wasted due to fragmentation. Compaction is required to remove fragmentation.

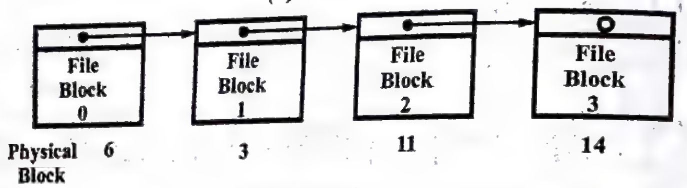
(ii) Linked Allocation Method – This method solves all problems of contiguous allocation. In linked allocation scheme each file is a linked list of disk blocks scattered on the disk. The first word of each block is used as a pointer to the next one and the rest of block is used for data. The directory entry contains a pointer to the first and last blocks of the file.

To create a new file, a new entry is created in the directory. Each directory entry has a pointer to the first disk block of the file. This pointer is initialized to nil to indicate an empty file. The size field is set to 0. A write to the file causes

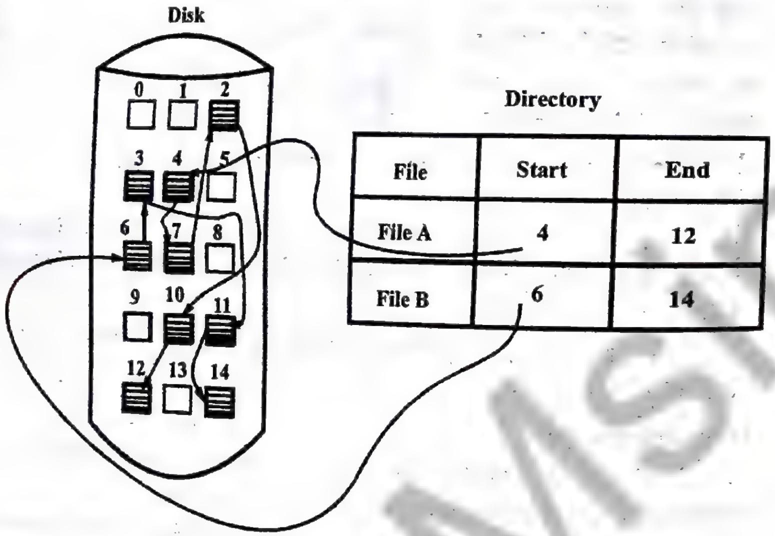
a free block to be found, and this new block is then written and linked to the end of the file. To read a file, read blocks by following the pointers from block to block.



(a) File A



(b) File B



(c) Linked Allocation of Disk Space

Fig. 2.5

Advantages –

(a) There is no wastage of memory, every block can be used.

(b) There is no external fragmentation only internal fragmentation is possible in last block.

(c) Directory needs only two values to remember file – starting block address and end block address.

(d) The file size does not need to be declared when that file is created.

Disadvantages –

(a) It can be used effectively only for sequential-access files but it is inefficient to support random access.

(b) Space is wasted in storing pointers.

(c) Access time, seek time and latency time are higher than contiguous allocation.

(iv) Indexed Allocation Method – Indexed allocation method eliminates the disadvantages of the linked list allocation by bringing all the pointers together into one location, called the index block.

Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. The directory contains the address of the index block as shown in fig. 2.6. To read the i^{th} block, use the pointer in the i^{th} index-block entry to find and read the desired block.

When the file is created, all pointers in the index block are set to nil. When the i^{th} block is first written, a block is obtained and its address is put in the i^{th} index-block entry.

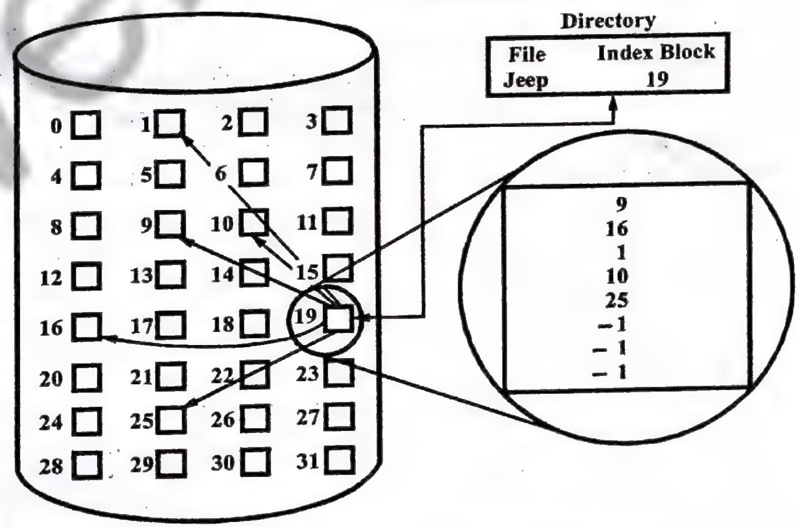


Fig. 2.6 Indexed Allocation of Disk Space

Advantages –

(a) The entire block is available for data using this method.

(b) Random access is much easier. Although the chain must be followed to find a given offset within the file.

(c) There is no external fragmentation using this method.

(d) Directory entry needs only to keep the starting block number of a file.

Disadvantages –

(a) The primary disadvantage is that the entire index table must be in memory all the time to make it work.

(b) It suffers from wasted space.

Q.16. Write characteristic of contiguous file allocation.

(R.G.P.V., Dec. 2014)

Ans. Some characteristics of contiguous file allocation are as follows –

(i) Pre-allocation is required.

(ii) It requires only single entry for a file.

(iii) It supports variable size portions.

(iv) Allocation frequency is only once.

DIRECTORY STRUCTURES, FILE PROTECTION, SYSTEM CALLS FOR FILE MANAGEMENT, DISK SCHEDULING ALGORITHMS

Q.17. Why directories are needed ?

Ans. The file systems of a computer can be very large. Some systems store millions of file on TBs (Tera bytes) of disk. To keep track of files, file system normally provides directories which in many systems, are themselves files. To manage all these data, we need to organize them. This organization usually done in two parts. First, disks are split into one or more partitions, also known as minidisks in the IBM world or volumes in the PC. Generally, each disk has at least one partition, which is a low-level structure in which files and directories reside. Sometimes partitions are used to provide several separate areas within one disk, each treated as a separate storage device, whereas other systems allow partitions to be larger than a disk to group disks into one logical structure. Thus, the user needs to be concerned with only the logical directory and file structure, and can avoid completely the problems of physically allocating space for files. For this reason partitions can be thought of as virtual disks. Partitions can also store multiple operating systems, allowing a system to boot and run more than one OS.

Second, each partition contains information about files within it. This information is kept in entries in a device directory or volume table of contents. The device directory records information – such as name, location, size, and type – for all files on that partition. Fig. 2.7 shows the typical file-system organization.

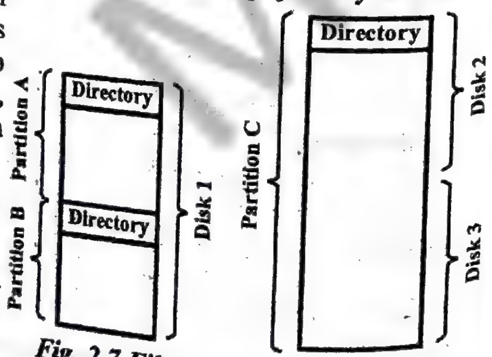


Fig. 2.7 File System Organization

The directory can be viewed as a symbol table that translates file names into their directory entries.

Q.18. What are the ways to implement directory ? Explain.

(R.G.P.V., Dec. 2010)

Ans. There are two ways to implement directory –

(i) Linear List – This method uses a linear list of file names with pointers to the data blocks. A linear list of directory entries requires a linear search to find a particular entry. This method is simple to program but time-consuming to execute. To create a new file, we must first search the directory to be sure that there is no file existing with the same name. If there is no file, then, we add a new entry at the end of the directory. To delete a file, we search the directory for the named file, then release the space allocated to it. To reuse the directory entry, we can do one of several things. First, we can mark the entry as unused. Second, we can attach it to a list of free directory entries. A third alternative is to copy the last entry in the directory into the freed location and to decrease the length of the directory. A linked list can also be used to decrease the time to delete a file.

The real disadvantage of a linear list of directory entries is the linear search to find a file. Directory information is used frequently and users would notice a slow implementation of access to it. In fact, many operating systems implement a software cache to store the most recently used directory information. A cache hit avoids constantly rereading the information from disk. A sorted list allows a binary search and decreases the average search time. However, the requirement that the list must be kept sorted may complicate creating and deleting files, since we have to move substantial amounts of directory information to maintain a sorted directory. A more sophisticated tree data structure, such as a B-tree, might help here. An advantage of the sorted list is that a sorted directory listing can be produced without a separate sort step.

(ii) Hash Table – In this method, a linear list stores the directory entries, but a hash data structure is also used. The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list. Therefore, it can greatly decrease the directory search time. Insertion and deletion are also fairly straightforward, although some provision must be made for *collisions* – situations where two file names hash to the same location. The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.

For example, assume that we make a linear-probing hash table that holds 64 entries. The hash function converts file names into integers from 0 to 63, for instance, by using the remainder of a division by 64. If we later try to create a 65th file, we must enlarge the directory hash table – say, to 128

entries. As a result, we need a new hash function that must map file names to the range 0 to 127 and we must reorganize the existing directory entries to reflect their new hash-function values. Alternatively, a chained-overflow hash table can be used. Each hash entry can be a linked list instead of an individual value and we can resolve collisions by adding the new entry to the linked list. Lookups may be somewhat slowed, because searching for a name might require stepping through a linked list of colliding table entries, but this is likely to be much faster than a linear search through the entire directory.

Q.19. What are the basic operations performed on a directory ?

Ans. There are various operations performed on directory they are as follows –

(i) CREATE – A directory is created. It is empty, except for dot and dotdot directories, which are put there automatically by the operating system.

(ii) DELETE – A directory is deleted. Only an empty directory can be deleted. A directory containing only dot and dotdot is considered empty as these usually cannot be deleted.

(iii) OPENDIR – Directories can be read, for example, to list all the files in a directory, a listing program opens the directory to read out the names of all the files it contains. Before a directory can be read, it must be opened.

(iv) CLOSEDIR – When a directory has been read, it should be closed to free up internal table space.

(v) LINK – Linking is a technique that allows a file to appear in more than one directory. This system call specifies an existing file to the name specified by the path. In this way, the same file may appear in multiple directories.

(vi) UNLINK – A directory entry is removed. If the file being unlinked is only present in one directory, it is removed from the file system. If it is present in multiple directories, only the path name specified is removed.

Q.20. Describe how a file directory system can be organized into one-level, two-level, and tree-structure directories. ***(R.G.P.V., Dec. 2016)***

Ans. The most common schemes for defining the logical structure of a directory are given below –

(i) One level/single-level directory
(ii) Two-level directory
(iii) Tree-structured directory.

(i) One-level/Single-level Directory – If system supports only a single directory and all files are stored in the same directory, then it is known as one-level directory. It is the simplest directory structure. However, a single-level directory has limitations, when the number of files increases or when the system

has more than one user. Since all files are in the same directory, they must have unique names. If two users choose the same file names, then the unique-name rule is violated. Thus, conflicts and confusion will quickly make the system unworkable.

Even a single user on a single-level directory may find it difficult to remember the names of all the files, as the number of files increases. Fig. 2.8 shows single-level directory structure.

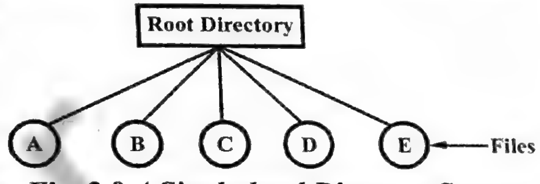


Fig. 2.8 A Single-level Directory System Containing Five Files Named A, B, C, D and E

(ii) Two-level Directory – In this type of directory structure, a different directory is given to a different user. Each of such directory is related to a user which has a group of files.

The files stored in same user's directory must have different names. But files in different user's directories may have same names. This type of file structure resolves the name collision problem.

Some systems do not support local user files to be accessed by other users to perform some cooperate task.

In fig. 2.9, two-level directory system is shown. It has three directories, those are allocated to users A, B and C. So user A can create files in directory A and user B can create files in directory B. Similarly user C can create files in directory C.

The two-level directory structure has disadvantages. This structure isolates one user from another. This isolation is an advantage when the users are completely independent, but is a disadvantage when the users want to cooperate on some task and to access one another's files.

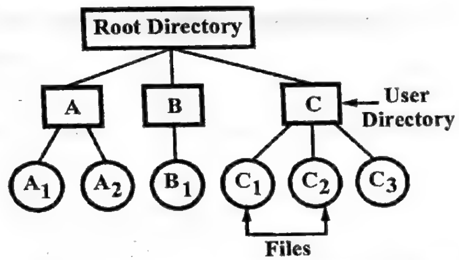


Fig. 2.9 A Two-level Directory System

(iii) Tree-structured Directory – The generalization of two-level directory structure is to extend the directory structure to a tree of arbitrary height. This generalization allows users to create their own subdirectories and to organize their files accordingly. The tree is the most common directory structure. The tree has a root directory. Every file in the system has a unique path name. A path name is the path from the root, through all subdirectories, to a specified file.

A directory (or subdirectory) contains a set of all files or subdirectories. A directory is also a file, but it is treated in a special way. All directories have the same internal format. One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).

Path names can be of two types – absolute path names or relative path names. An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name is a path from the current directory.

Fig. 2.10 shows a tree-structured directory system.

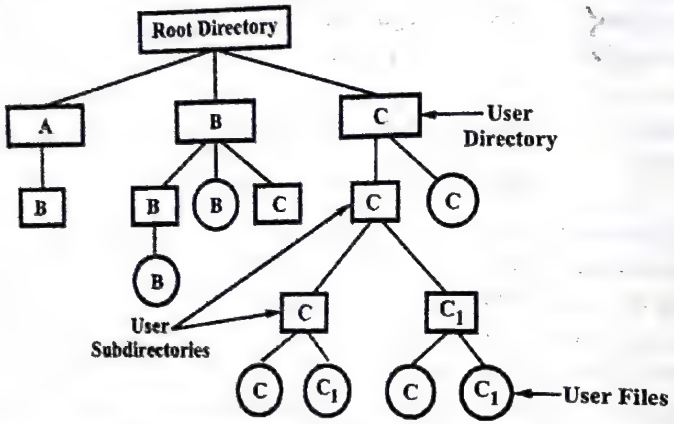


Fig. 2.10 A Tree-structured Directory System

Q.21. Discuss in brief about acyclic graph directories.

(R.G.P.V., Dec. 2002)

Or

Discuss about acyclic graph directories.

(R.G.P.V., June 2007, Nov./Dec. 2007)

Or

Discuss the directory structure which allows sharing of files or directories.

(R.G.P.V., Nov./Dec. 2007)

Ans. Acyclic graph directory structure allows to share files or directories among users. This is graph like structure where no cycle exists. The concept of file sharing is different from file duplication. In file duplication, each user has copy of original file and he can change it's contents, but by changing the contents of a copy held by a user will not reflect any change in other copies. On the other hand, in file sharing, any modification done by a user in shared file, immediately reflects to other users of this shared file.

The tree structure does not allow to share files or directories, but an acyclic graph allows to share files or directories. So, same files or directories can appear in different directories. An acyclic graph file system is shown in fig. 2.11.

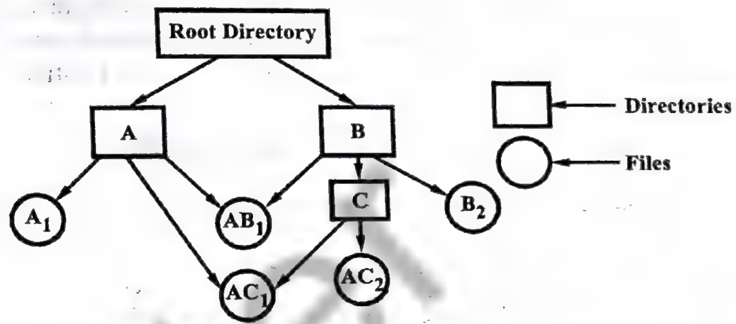


Fig. 2.11 Acyclic-graph Directory System

UNIX implements acyclic graph directory system using link concept. A link the pointer to the original file or subdirectory. When a link is created by using other name for a file, first the original file name entry is searched in the directory, then an entry is made for link with the address of original file. This approach provides more flexibility than tree-structure directory system but it is also more complex.

The problems with acyclic graph directory system are as follows – first, a file may have more than one absolute address. Second is the problem of deletion. If a user wants to delete shared file, then after deletion other user will point to garbage value, that contain dangling pointers.

One way to resolve this deletion problem in UNIX system, that uses links to share file, is to remain original copy until it is forcely not deleted by user. If a link is deleted, then only corresponding entry is removed from directory. But, if file is deleted then with deletion of entry from directory, space is also deallocated. After that, all dangling links are removed by searching.

Because of these problems associated with acyclic graph directory system, MS-DOS uses tree-structure directory system.

Q.22. Write short note on file protection mechanism.

Ans. The need to protect files is a result of the ability to access files. Systems that do not permit access to the files of other users do not need protection. Thus, we can provide protection by prohibiting access. Alternatively, we can provide free access with no protection. Hence, controlled access is needed for both approaches.

Protection mechanisms provide controlled access by limiting the types file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. The types of operations that may be controlled, are read, write, execute, append, delete, list. Other operations, such as renaming, copying, or editing the file, may also be controlled.

The most common approach to the protection problem is to make access dependent on the identity of the user. The scheme is to implement identity –

dependent access to associate with each file and directory an ***access-control list*** (ACL) specifying the user name and the types of access allowed for each user. When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the request access, the access is allowed. Otherwise, a protection violation occurs, and the user is denied access to the file.

Another approach to protection problem is to associate a password with each file. Just as access to the computer system is controlled by a password, access to each file is controlled by a password.

Q.23. How multiple users are allowed to share files ? Explain the possible methods of file sharing.

Ans. File sharing is very desirable for users who want to collaborate and to reduce the effort required to achieve a computing goal. Some of the difficulties involved in allowing users to share files. Therefore, user-oriented operating systems must accommodate the need to share files in spite of the inherent difficulties.

When an operating system accommodates multiple users, the issues of file sharing, file naming and file protection become preeminent. Given a directory structure that allows files to be shared by users, the system must mediate the file sharing. The system either can allow a user to access the files of other users by default, or it may require that a user specifically grant access to the files.

To implement sharing and protection, the system must maintain more file and directory attributes than on a single-user system. Most systems have evolved to the concepts of file/directory *owner* (or *user*) and *group*. The owner is the user who may change attributes, grant access and has the most control over the file or directory. The group attribute of a file is used to define a subset of users who may share access to the file.

Most systems implement owner attributes by managing a list of user names and associated user identifiers (user IDs). These numerical IDs are unique for each user. When a user logs in to the system, the authentication stage determines the appropriate user ID for the user.

That user ID is associated with all of the user's processes and threads. When they need to be user readable, they are translated back to the user name via the user name list. Likewise, group functionality can be implemented as a system-wide list of group names and group identifiers. Every user can be in one or more groups, depending upon operating system design decisions. The user's group IDs are also included in every associated process and thread. The owner and group IDs of a given file or directory are stored with the other file attributes. When a user requests an operation on a file, the user ID can be compared to the owner attribute to determine if the requesting user is the owner of the file. Likewise, the group IDs can be compared. The result indicates which permissions are applicable. The system then applies those permissions to the requested operation and allows or denies it.

Many systems have multiple local file systems, including partitions of a single disk or multiple partitions on multiple attached disks. In these cases, the ID checking and permission matching are straightforward, once the file systems are mounted.

Q.24. How can you provide protection to a file when it is shared among several users ? ***(R.G.P.V., June 2016)***

Ans. The protection to a file is required in multiuser environment where a file is shared among several users. Protection mechanism must provide controlled access by restricting the types of files which can be made. Access is permitted or denied depending upon several factors, one of which is the type of access requested. Several different types of operations may be controlled. Some of them are reading from file, writing on file, executing the file by loading into main memory, writing new information at the end of a file, deleting the file and releasing the space.

Many different protection mechanisms have been proposed. Each one has got advantages and disadvantages and must be selected which is suitable for its intended application. A large computer system needs different types of protection mechanisms compared to a small computer system used by a small group. The protection can be supported with file itself but the more common scheme is to provide protection with path. Thus, if a path name refers to a file in a directory, the user must be allowed access to both the directory and the file. In systems where files may have numerous path names, a given user may then have different access rights to a file depending upon the path name used. Some mechanisms to provide protection to a file are as follows –

(i) Passwords – In this approach, password is associated with each file. Just as a password is required to access a computer system, access to each file will be also controlled by a password. However, there are several disadvantages to this scheme. If we associate a separate password with each file, the number of passwords that need to be remembered are quite large, making the scheme impractical. If only one password is used for all the files, then once it is discovered all files are accessible. Some systems allow a user to associate a password with a subdirectory rather than an individual file to deal with this problem.

(ii) Access Lists – Another approach is to make access dependent on the identity of the user. Various users may need different types of access to a file or directory. An access list can be associated with each file and directory, specifying the user name and the types of access allowed for each user. When a user requests access to a particular file, the operating system checks the access list associated with each file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs and the user job is denied access to the file.

(iii) Access Groups – The main problem with access lists can be their length. If we want to allow everyone to read a file, we must list all users

with read access. Constructing such a list may be tedious and unrewarding task. The directory entry which previously was of fixed size needs now to be of variable size, resulting in space management more complicated. These problems can be resolved by using a condensed version of the access list. To condense the length of the access list, many systems recognize three classifications of users in connection with each file –

(a) Owner – The user who created the file.

(b) Group – A set of users who are sharing the file and need similar access.

(c) Universe – All other users in the system.

With this classification, only three fields are needed to define protection. Each field is often a collection of bits which either allow or prevent the access associated with that bits.

Q.25. Explain security threats and intruders.

Ans. **Security Threats** – There are three main goal of security. The first is to keep data confidential. It means the owner of this data want to keep it secret or want to give access to only some authorized users and hide from others.

The second is to maintain data integrity. It means unauthorized users should not be able to alter the data. It means, cannot be able to modify, delete or add the data.

The third is system availability. It means system is always available. One cannot make it unusable.

So security threats are the threats to break these security goals. The threat corresponding to keep data confidential is exposure of data. Similarly, for maintaining data integrity is tempering with data and for system availability is denial of service.

Privacy, is another problem that requires to keep one's information hide from others. Since every organisation has its some confidential data and maintained by general manager so general manager want to keep this data hidden from others, so that policies and plans of that organization should be private.

Intruders – Intruders are those people who want to read or write or modify data on which they have no access authorization. There are two types of intruders –

(i) Passive Intruder – One that only read the data without read authorization on that data, called passive intruder.

(ii) Active Intruder – One that can modify the data without write authorization on that data called active intruder.

Whenever a system is designed, it is kept in mind that from what kinds of intruders, it should be protected.

There are following categories of intruders –

(i) Non-technical Users – It is the tendency of human mind to find new thing. When a non-technical user connects to the shared file sever want to read files of others also, such are non-technical intruders. For such users some security mechanism is necessary.

(ii) Technical Users – Efficient programmers, students, operators want invention in this field so they always try to do new things. Intentionally they put efforts to break security.

(iii) Theft Intruders – Some programmers want to steal money from banks and industries by changing the data kept in computers. These type of intruders are called computer theft.

(iv) Market Competitors – There are a lot of competitors in same kind of business, so one want to steal programs, secrets, ideas, technology, design, business plans and so on of others firm. These are market competitor intruders.

Since each intruder works differently, so it is necessary to keep in mind the type of intruder against the security is maintained.

Q.26. What do you mean by authentication ? Explain in detail.

Ans. A major security problem for operating systems is authentication. The protection system depends on ability to identify the programs and processes currently executing. This ability, in turn, rests on our power to identify each user of the system. A user normally identifies himself.

The primary goal of authentication is to allow access to legitimate system users and to deny access to unauthorized parties. The two primary measures of authentication effectiveness are –

(i) The false acceptance ratio, that is, the percentage of legitimate users erroneously admitted, and

(ii) The false rejection ratio, that is, the percentage of legitimate users who are denied access due to failure of the authentication mechanism.

The main objective is to minimize both the false acceptance and the false rejection ratios.

One way authentication is usually based on –

(i) Passwords
(ii) Artifact-based authentication
(iii) Biometric techniques.

(i) Passwords – The most common approach to authenticating a user identify is the use of passwords. When the user identifies herself by user ID or account name, she is asked for a password. If the user-supplied password matches the password stored in the system, the system assumes that the user is legitimate.

Passwords are popular because they require no special hardware and are relatively easy to implement. On the negative side, passwords offer limited protection, as they may be relatively easy to obtain or guess. Unencrypted password files stored in a system are obviously an easy prey. Browsing and searching of waste may also turn up some passwords exchanged by users via electronic mail. User-chosen passwords are frequently dictionary words or proper names. This makes them easy to remember but also easy to guess. For example, user IDs, names or surnames, and their backword spelling typically account for a significant percentage of passwords. Password attacks usually first try these, and then perhaps a customized list of known common passwords. If these fail, the attacker may proceed with the trial-and-error of the words in online dictionaries. Password words from a dictionary facilitate breaking of the encryption key by exhaustive trial-and-error with the aid of computer. Whereas, a system-chosen password is usually random combinations of letters and numbers that are hard to guess but also hard to remember. As a result, the users tend to write them down and store them in a handy place near the terminal. This replaces exposure to guessing with exposure to revelation of the secret.

(ii) Artifact-based Authentication – The artifacts are generally used for user authentication include machine-readable badges and various incarnations of the electronic smart cards. Card readers may be installed in or near the terminals, and users are required to supply the artifact for authentication.

In many systems, artifact identification is coupled with the use of a password. That is, the user must insert the card and then supply his or her password. This form of authentication is common with automated teller banking machines. The artifact-based systems work especially well in environments where the artifact is also used for other purposes.

For example, in some companies badges are required for employees to gain access to work premises. The use of such a badge as an artifact for computer access and authentication can reduce the likelihood of the undetected loss on an artifact. Smart cards can augment this scheme by keeping even the user's password secret from the system. The unique user password is stored in an unreadable form within the card itself, which allows authentication without storage of passwords in the computer system. This makes it more difficult for perpetrators to uncover user passwords.

(iii) Biometric Techniques – The third major group of authentication mechanisms is based on the unique characteristics of each user. Some user characteristics can be relatively unobtrusively established by means of biometric techniques. These fall into two basic categories –

(a) Physiological characteristics, such as fingerprints, capillary patterns in retina, hand geometry, and facial characteristics.

(b) Behavioral characteristics, such as signature dynamics, voice pattern, and timing of keystrokes.

Generally, the behavioural characteristics, can vary with a user's state and thus may be susceptible to higher false acceptance or rejection rates. The detection devices are usually self-contained and independent of the computer system, which increases their resistance to common computer penetration methods and improves the potential for tamper-proofing. The drawbacks of biometric authentication include increased cost, potential invasion of privacy, and reductance of some users.

Q.27. Explain protection domains mechanism of file system.

Ans. A computer system may be viewed as consisting of a set of objects such as processes, that operate on and manipulate a set of objects. Objects include both hardware such as peripheral devices and memory segments, and software objects, such as files, arrays and semaphores.

From the software point of view, each object is an abstract data type. Each object has a unique name that differentiates it from all other objects in the system, and each can be accessed only through well-defined operations. For example, a CPU can only be executed on. Memory segments can only be read and written.

It is obvious that a way is needed to prohibit processes from accessing objects that they are not authorized to access. Furthermore, this mechanism must also make it possible to restrict processes to a subset of the legal operations when that is needed.

Some of these relationships may be expressed by means of an abstraction called protection domain, which specifies a set of objects and the types of operations that may be performed on each object. A protection domain is a collection of access rights, each of which is a pair < object identifier, rights set >.

Domains do not need to be disjoint, they may share access rights. Fig. 2.12 shows three domains, showing the objects in each domain and the rights [Read, Write, Execute] available on each object. The access right < Printer 1, {Write} > is shared by domain 2 and domain 3.

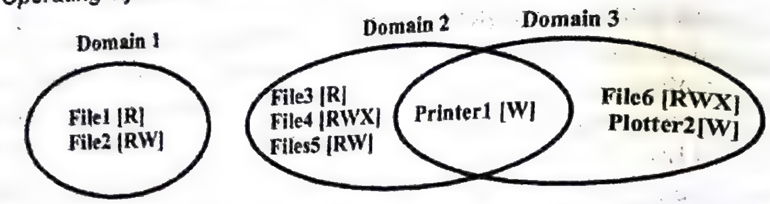


Fig. 2.12 System with Three Protection Domains

At every instant of time, each process runs in some protection domain. In other words, there is some collection of objects it can access, and for each object it has some set of rights. Processes can also switch from domain to domain during execution.

These relationships may be represented by means of an access matrix, which is representation of all access rights of all subjects to all objects in a computer system. It is usually depicted as a two-dimensional matrix, with protection domains as rows and system objects as columns. Both hardware and software objects are included in the access matrix.

Fig. 2.13 shows a small access matrix. Blank entries indicate no access rights. A process executing in domain D2 can access only one object file 2 in read-only mode. File 3 is presumably a shared utility that is maintained by domain D3 and is also executable in domain D1.

Although a useful model, access matrices are inefficient for storage of access-rights in a computer system because they are large and sparse.

Domain \ Object	File 1	File 2	File 3	Printer
D1	Read Write		Execute	Output
D2		Read		
D3			Read Write Execute Copy	Output

Fig. 2.13 Access Matrix

Q.28. Discuss the key features of NTFS file system.

Or

Write short note on key features of windows file system.

(R.G.P.V., Dec. 2016)

Ans. NTFS is a flexible and powerful file system built, on an elegantly simple file system model. The most noteworthy features of NTFS include the following –

(i) Recoverability – High on the list of requirements for the new W2K file system was the ability to recover from system crashes and disk failures. In the event of such failures, NTFS is able to reconstruct disk volumes

and return them to a consistent state. It does this by using a transaction processing model for changes to the file system; each significant change is treated as an atomic action that is either entirely performed or not performed at all. Each transaction that was in process at the time of a failure is subsequently backed out or brought to completion. In addition, NTFS uses redundant storage for critical file system data, so that failure of a disk sector does not cause the loss of data describing the structure and status of the file system.

(ii) Security – NTFS uses the W2K object model to enforce security. An open file is implemented as a file object with a security descriptor that defines its security attributes.

(iii) Large Disks and Large Files – NTFS supports very large disks and very large files more efficiently than most other file systems, including FAT.

(iv) Multiple Data Streams – The actual contents of a file are treated as a stream of bytes. In NTFS it is possible to define multiple data streams for a single file. An example of the utility of this feature is that it allows W2K to be used by remote Macintosh systems to store and retrieve files. On Macintosh, each file has two components – the file data and a resource fork that contains information about the file. NTFS treats these two components as two data streams.

(v) General Indexing Facility – NTFS associates a collection of attributes with each file. The set of file descriptions in the file management system is organized as a relational database, so that files can be indexed by any attribute.

Q.29. Describe NTFS volume and file structure.

Ans. NTFS makes use of the following disk storage concepts –

(i) Sector – The smallest physical storage unit on the disk. The data size in bytes is a power of 2 and is almost always 512 bytes.

(ii) Cluster – One or more contiguous (next to each other on the same track) sectors. The cluster size in sectors is a power of 2.

(iii) Volume – A logical partition on a disk, consisting of one or more clusters and used by a file system to allocate space. At any time, a volume consists of a file system information, a collection of files, and any additional unallocated space remaining on the volume that can be allocated to files. A volume can be all or a portion of a single disk or it can extend across multiple disks. If hardware or software RAID 5 is employed, a volume consists of stripes spanning multiple disks. The maximum volume size for NTFS is 2^{64} bytes.

The cluster is the fundamental unit of allocation in NTFS, which does not recognize sectors. For example, suppose each sector is 512 bytes and the system is configured with two sectors per cluster (one cluster = 1K bytes). If a user creates a file of 1600 bytes, two clusters are allocated to the file. Later, if the user updates the file to 3200 bytes, another two clusters are allocated.

The clusters allocated to a file need not be contiguous; it is permissible to fragment a file on the disk. Currently, the maximum file size supported by NTFS is 2^{32} clusters, which is equivalent to a maximum of 2^{48} bytes.

The use of clusters for allocation makes NTFS independent of physical sector size. This enables NTFS to support easily nonstandard disks that do not have 512-byte sector size, and to support efficiently very large disks and very large files by using a larger cluster size. The efficiency comes from the fact that the file system must keep track of each cluster allocated to each file; with larger clusters, there are fewer items to manage.

Table 2.2 shows the default cluster sizes for NTFS. The defaults depends on the size of the volume. The cluster size that is used for a particular volume is established by NTFS when the user requests that a volume be formatted.

Table 2.2 Windows NTFS Partition and Cluster Sizes

Volume Size	*Sectors Per Cluster*	*Cluster Size*
≤ 512 Mbyte	1	512 bytes
512 Mbyte-1 Gbyte	2	1K
1 Gbyte-2 Gbyte	4	2K
2 Gbyte-4 Gbyte	8	4K
4 Gbyte-8 Gbyte	16	8K
8 Gbyte-16 Gbyte	32	16K
16 Gbyte-32 Gbyte	64	32K
> 32 Gbyte	128	64K

Q.30. Explain the file allocation table (FAT) in the context of linked allocation.

Ans. The use of a file allocation table (FAT) in linked allocation provides important variation in the performance of allocation of disk space. This simple but efficient method of disk-space allocation is implemented in the MS-DOS and OS/2 operating systems. A section of disk at the starting of each partition is set aside to contain the table. The table contains one entry for each disk block and is indexed by block number. The FAT is much like a linked list. The directory entry used to contain the block number of the first file. The table entry indexed by start block number that contains pointer of the next block in the file. A block in this way, follows the concept of link-list. At the last of these link-list, a special end-of-file value as the table entry is used. Block, that have not been used, is indicated by a 0 table value. And a separated available list is used in the file allocation table (FAT). For allocating a new block to a file, there is the first 0-valued table entry is provided to the process and new block further point to end-of-file. Fig. 2.14 shows an example using file allocation table, in which file consists of disk block 225, 518 and 423.

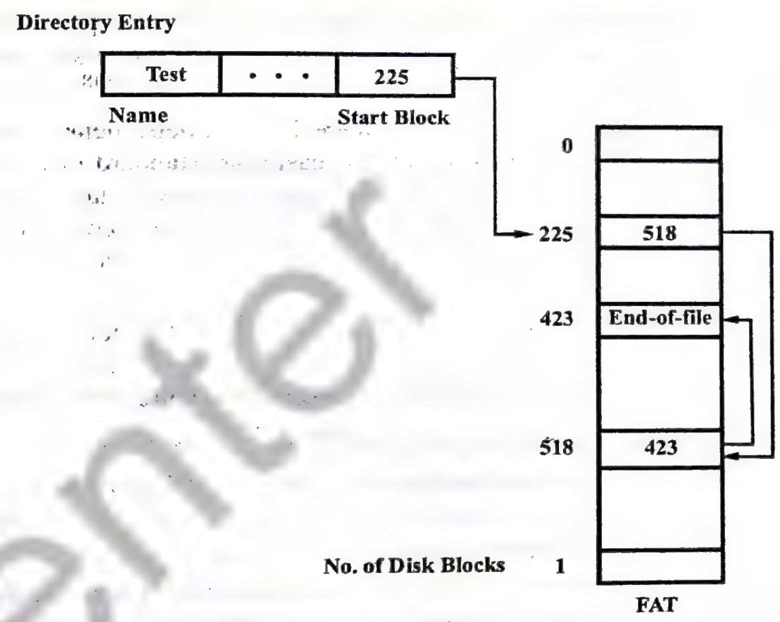


Fig. 2.14 File Allocation Table

Q.31. Write any five advantages and five disadvantages of FAT-32 file system.

Ans. FAT-32 has the following advantages –

(i) The root folder on a FAT-32 drive is an ordinary cluster chain and can be located anywhere on the volume. For this reason, FAT-32 does not restrict the number of entries in the root folder.

(ii) FAT-32 uses smaller clusters (4 KB for volumes upto 8 GB), so it allocates disk space more efficiently than FAT-16. Depending on the size of your files, FAT-32 creates the potential for tens and even hundreds of megabytes of additional free disk space on larger volumes compared to FAT-16.

(iii) FAT-32 can automatically use the backup copy of the file allocation table instead of the default copy (with FAT-16, only a disk repair tool such as Chkdsk can implement the backup).

(iv) The boot sector is automatically backed up at a specified location on the volume, so FAT-32 volumes are less susceptible to single points of failure than FAT-16 volumes.

(v) FAT-32 is best for cross-compatibility with other platforms. FAT-32 is more robust. FAT-32 also reduces the resources necessary for the computer to operate.

FAT-32 has the following disadvantages –

(i) The largest FAT-32 volume that Windows 2000 can format is 32 GB.

(ii) FAT-32 volumes are not directly accessible from operating systems other than Windows 95 OSR2 and Windows 98.

(iii) If you have a startup failure, you cannot start the computer by using an MS-DOS or Windows 95 (excluding version OSR2 and later) bootable floppy disk.

(iv) There is no built-in file system security or compression scheme with FAT-32.

(v) More than one identical copies of FAT are maintained for protection.

Q.32. Write short note on Linux PROC file system.

Ans. The Linux **process file system,** known as the ***proc*** file system, is an example of a file system whose contents are not actually stored anywhere but are computed on demand according to user file I/O requests.

A proc file system is not unique to Linux. SVR 4 Unix introduced a proc file system as an efficient interface to the kernel's process debugging support – Each subdirectory of the file system corresponded not to a directory on any disk, but rather to an active process on the current system. A listing of the file system reveals one directory per process, with the directory name being the ASCII decimal representation of the process' unique process identifier (PID).

Linux implements such a proc file system, but extends it greatly by adding a number of extra directories and text files under the file system's root directory. These new entries correspond to various statistics about the kernel and the associated loaded drivers. The proc file system provides a way for programs to access this information as plain text files, which the standard Unix user environment provides powerful tools to process.

The proc file system must implement two things – a directory structure and the file contents within. The proc file system defines a unique and persistent inode number for each directory and the associated files. Once such a mapping exists, it can use this i-node number to identify what operation is required when a user tries to read from a particular file i-node or to perform a lookup in a particular directory i-node. When data are read from one of these files, the proc file system will collect the appropriate information, format it into textual form and place it into the requesting process' read buffer.

The mapping from i-node number to information type splits the i-node number into two fields. In Linux, a PID is 16 bits wide, but an i-node number is 32 bits. The top 16 bits of the i-node number are interpreted as a PID and the remaining bits define the type of requested information about that process.

A PID of zero is not valid, so a zero PID field in the i-node number means that this i-node contains global information rather than process-specific. Separate global files exist in proc to report information such as the kernel version, free memory, performance statistics and currently running drivers.

Q.33. Describe the Linux virtual file system in detail.

Ans. The Linux virtual file system (VFS) is designed around object-oriented principles. It has two components – a set of definitions that define what a file object is allowed to look like, and a layer of software to manipulate those objects. The three main object types defined by VFS are the ***i-node-object*** and the ***file-object*** structures, which represent individual files, and the ***file-system-object***, which represents an entire file system.

The VFS defines a set of operations for each of these three types of objects, that must be implemented by that structure. Every object of one of these types contains a pointer to a function table. The function table lists the addresses of the actual functions that implement those operations for that particular object. Thus, the VFS software layer can perform an operation on one of these objects by calling the appropriate function from that object's functions table.

The file system object represents a connected set of files that forms a self-contained directory hierarchy. The operating system kernel maintains a single file system object for each disk device mounted as a file system and for each networked file system currently connected. The file system object's main responsibility is to give access to i-nodes. The VFS identifies every i-node by a unique (file system inode number) pair, and it finds the inode corresponding to a particular i-node number by asking the file system object to return the i-node with that number.

The i-node and file-objects are the mechanisms used to access files. An i-node-object represents the file as a whole, and a file-object represents a point of access to the data in the file. A process cannot access an i-node's data contents without first obtaining a file-object pointing to the i-node. The file object keeps track of where in the file the process is currently reading or writing, to keep track of sequential I/O. It also remembers whether the process asked for write permissions when the file was opened, and keeps track of the process' activity if necessary to perform adaptive read-head, fetching file data into memory in advance of the process requesting it, to improve performance.

File objects belong to a single process, but i-node-objects do not. Even once a file is no longer being used by any processes, its i-node-object may still be cached by the VFS to improve performance if the file is used again in near future.

Q.34. Compare file system in Windows and Linux.

(R.G.P.V., June 2010, Dec. 2015)

Ans. **File System in Linux** – Linux supports over 12 file systems, with the technology of NFS. When Linux (i.e., the operating system code) is linked, the choice of the default file system needs to be specified. The other file systems can be called dynamically, depending on the requirements. The Ext2 file system is the most popular choice. It is similar to the Berkley file system.

This file system considers that the disk begins with a boot block and then rest of the disk to be made up of a series of ***block groups***. Block groups are numbered sequentially, and they contain many sub-fields. The overall organization is shown in fig. 2.15.

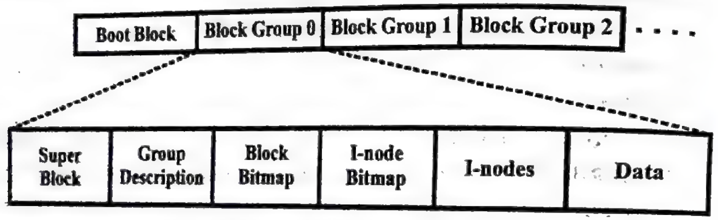


Fig. 2.15 Linux File System (Ext2)

The details of the sub-fields in a ***block group*** are as follows –

(i) Super Block – Specifies the number of blocks, the number of i-nodes, block size etc.

(ii) Group Description – Contains information about the bitmap location, number of free blocks, i-nodes, directories in the group, etc.

(iii) Block Bitmap – Contains a list of free blocks.

(iv) I-node Bitmap – Contains a list of free i-nodes.

(v) I-nodes – These are the actual i-nodes. Each i-node is 128 byte long.

Linux supports the following file types –

(i) Directory – This is simply a list of file names.

(ii) Ordinary File – This is a file containing data or application program or executable.

(iii) Symbolic Link – This file is actually a link to (or path of) another file.

(iv) Special File – This refers to a device driver.

(v) Named Pipe – This is a common channel between two or more processes for data exchange.

File System in Windows – Windows 2000 (W2K) supports a number of file systems including the file allocation table (FAT) that runs on Windows 95, MS-DOS and OS/2. But the developers of W2K also designed a new file system, the W2K File System (NTFS), that is intended to meet high-end requirements for workstations and servers. Examples of high-end applications include the following –

(i) Client/server applications such as file servers, computer servers and database servers.

(ii) Resource-intensive engineering and scientific applications.

(iii) Network applications for large corporate systems.

Key Features of NTFS – Refer Q.28.

NTFS Volume and File Structure – Refer Q.29.

Q.35. Describe i-node with its fields in brief.

Or

Write short note on i-node. ***(R.G.P.V., May 2018)***

Ans. An i-node is a little table associated with each file on the disk. Every file has a unique i-node. An i-node is a record that describes the attributes of a file including the layout of its data on disk. I-nodes exist in a static form on disk. The i-node contains the information necessary for a process to access a file, such as file ownership, access rights, file size and location of the file's data in the file system. Disk i-nodes consist of the following fields –

(i) File Owner Identifier – File ownership is divided between an individual owner and a group owner and define the set of users who have access rights to a file. There supervisor or superuser has access rights to all files in the system.

(ii) File Type – Files may be of different types like regular directory, character special, block special or FIFO (pipes).

(iii) File Access Permission – The protection of files is done according to three classes – the owner, the group owner and other users. Each class has access rights to read, write and execute the file which can be set individually.

(iv) File Access Times – It gives the time the file was last modified, when it was last accessed and when the i-node was last modified.

(v) Number of Links to the File – It gives the number of names the file has in the directory hierarchy.

(vi) Table of Contents for the Disk Addresses of Data in a File – Although users treat the data in a file as a logical stream of bytes, the kernel saves the data in discontiguous disk blocks. The inode i-dentifies the disk blocks that contain the file's data.

(vii) File Size – Data in a file is addressable by the number of bytes from the beginning of that file the starts from byte offset 0 and gives the file size 1 greater than the highest byte offset of data in the file.

Q.36. Write short notes on – **(R.G.P.V., June 2017)**
(i) FAT (ii) I-node

Ans. (i) *FAT* – Refer Q.30.

(ii) *I-node* – Refer Q.35.

Q.37. Discuss various file access methods. **(R.G.P.V., Dec. 2011)**

Or

Write short note on file access methods. **(R.G.P.V., June 2011)**

Or

Explain various methods of accessing file with examples. **(R.G.P.V., Dec. 2015)**

Explain various file access methods. **(R.G.P.V., May 2018)**

Ans. The information in the file can be accessed in several ways –

(i) ***Sequential Access Method*** – In this method, a process can read all the bytes or records in a file in order, starting at the beginning, but cannot skip around and read them out of order.

The bulk of operations on a file is reads and writes. A read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, a write appends to the end of the file and advances to the end of the newly written material. Sequential files are convenient when the storage medium is magnetic tape rather than disk. Sequential access is depicted in fig. 2.16.

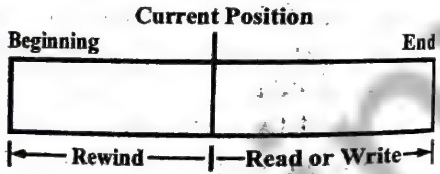


Fig. 2.16 Sequential Access File

(ii) ***Direct/random/relative Access Method*** – A direct access file is made up of collection of records, each of which is identified by a number called ***record number.***

In direct access file, one can access any record at any time for reading or writing. For example, records can be read as follows, first read record 5, then record 58, then record 1, that has no order of reading and writing to a file. One can directly access to any number of records at once.

The user specifies relative block numbers of file to the operating system. The relative block numbers are started from 0 to the total number of blocks in file. On other hand, absolute block numbers are started by the first block number of the file.

The direct access provides fastest way to access a record of file by first specifying the record number.

(iii) ***Index Sequential Access Method*** – This method is developed by slight modification in the direct access method. The index contains pointer

to the various blocks or records. To find an entry in the file, we first search the index and find the pointer of the record to access file entry directly.

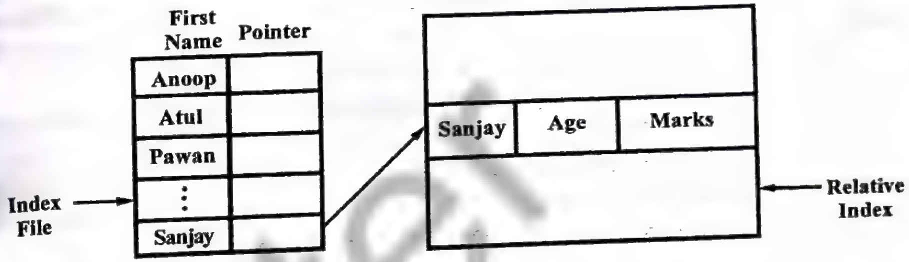


Fig. 2.17 ISAM

Q.38. Discuss various file allocation and access methods. Compare their advantages and disadvantages. **(R.G.P.V., Dec. 2016)**

Ans. Refer Q.15 and Q.37.

Q.39. Explain the three levels of device abstraction and disk storage addressing method identifiable in implementations of the file-management system.

Ans. (i) ***File-relative Logical Addressing*** – The storage system is viewed as a collection of named files, at the highest level of abstraction. The items stored in secondary storage are in the form of filename or offset are addressed in the form of two-component. This form of addressing are used by most of the applications and file-related system calls.

(ii) ***Volume-relative Logical Addressing*** – An abstraction of the disk is provided by the disk device drivers as a linear array of sectors. The independent devices of the file system use this form of addressing, (sector, offset). Devices such as small computer system interface provide this form of disk abstraction directly by means of hardware controllers integrated in the drives themselves.

(iii) ***Drive-relative Physical Addressing*** – The three main components was used by this level in the form of physical addresses such as cylinder, head, sector. It was described that this level of addressing is obtained by software device drivers that translate it into the actual drive-control signals.

Q.40. Write short note on disk scheduling.

Ans. In multiprogrammed computing systems, many processes may be generating requests for reading and writing disk records. Because these processes sometimes make requests faster than they can be serviced by the moving-head disks, waiting lines or queues build up for each device. Some computing systems simply service these requests on a first-come-first-served basis. Whichever request for service arrives first is serviced first. FCFS is a fair method of allocating service, but when the request rate becomes heavy,

FCFS can result in every long waiting times. FCFS exhibits a random se pattern in which successive requests can cause time-consuming seeks fro the innermost to the outermost cylinders. To minimize the time spent in seekin records, it seems reasonable to order the request queue in some manner oth than FCFS. This process is called ***disk scheduling.***

Disk scheduling involves a careful examination of pending requests determine the most efficient way to service the requests. A disk schedul examines the positional relationships among waiting requests. The reque queue is then reordered so that the requests will be serviced with minimu mechanical motion. The most common disk scheduling policies are –

(i) FCFS (first-come-first-serve) scheduling
(ii) SSTF (shortest-seek-time-first) scheduling
(iii) SCAN Scheduling
(iv) Circular SCAN (C-SCAN) scheduling
(v) LOOK scheduling
(vi) C-LOOK scheduling.

Q.41. Describe the selection criteria of a disk-scheduling algorithm.

Ans. Selection of disk-scheduling algorithm depends on the specificatio of system, because each algorithm has its own merit and demerit. On basis o suitability and specifications, disk-scheduling algorithm are chosen. For exampl – SSTF (shortest-seek-time-first) algorithm is common and has a natural appeal it maximizes performance over FCFS. Systems which are suffering from heavy load, SCAN and C-SCAN are used to provide better performance. At certain extent, it also minimizes the possibility of starvation problem. Some basic criteria of selections are as follows –

(i) The performance of scheduling algorithm depends heavily on the number and types of request. For example, assume that queue usually has just one outstanding request. Then, all scheduling algorithm are formed to behave the same way.

(ii) The file-allocation method influences greatly the request for disk service. A program fetching a contiguously allocated file will produce several requests, that are close together on the disk, outcoming in limited head movement.

(iii) The disk-scheduling should be chosen in a way as to provide a separate module of the operating system. So that it can be replaced with a different algorithm if necessary.

(iv) The memory location of directories and index blocks is also important factors in selection of a disk-scheduling algorithms. Since every file must be opened to be used, and opening a file needs searching the directory structure, the directories will be accessed frequently. For example, suppose that a directory entry is on the first cylinder and file's data are on the last

cylinder. In such situation, the disk head will move around the entire width of the disk. So the scheduling should be chosen in a way to reduce the disk-arm movement, particularly for read requests.

Q.42. Explain the first-come-first-serve (FCFS) type disk sheduling.

Ans. The simplest form of disk scheduling is the first-come-first-served (FCFS) algorithm. It means that the items are processed from the queue in sequential order. This algorithm has the advantage of being fair in the sense that once a request has arrived, its place in the schedule is fixed. A request cannot be displaced because of the arrival of a higher priority request. However, this algorithm does not provide the fastest service.

For example, the requests for I/O to blocks are on following cylinders –

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

Let disk drive has 5000 cylinders numbered from 0 to 4999. The disk head is initially at 143. It will first move from 143 to 86, then to 1470, 913, 1774, 948, 1509, 1022, 1750 and at last to 130. This is shown in fig. 2.18.

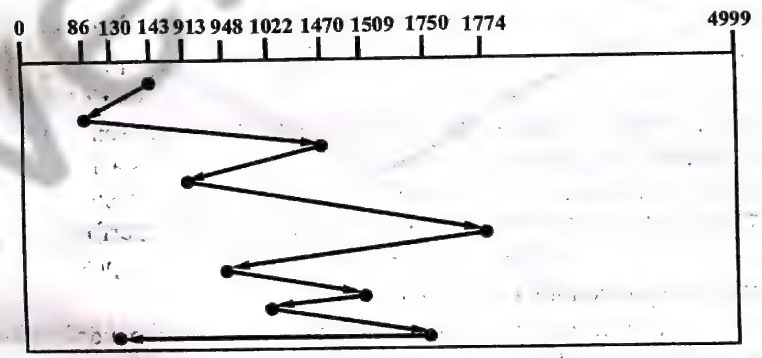


Fig. 2.18 FCFS Disk Scheduling

Total head movement is obtained by adding the head movements one by one. So, total head movements

$$= 57 + 1384 + 557 + 861 + 826 + 561 + 487 + 728 + 1620$$
$$= 7081 \text{ cylinders}$$

The problem with this scheduling is to swing from 913 to 1774 and then to 948. If 913 and 948 both requests are served together one after another then total head movement can be decreased and performance can be improved.

Q.43. Describe about shortest-seek-time-first (SSTF) type disk scheduling.

Ans. SSTF is reasonable to service all the requests close to the current head position, before moving the head far away to service other requests.

This assumption is the basis for the shortest-seek-time-first (SSTF) algorith. Thus, SSTF policy is to select the disk I/O request that requires the le movement of the disk arm from its position to incur the minimum seek tim Of course, always choosing the minimum seek time does not guarantee th the average seek time over a number of arm movements will be minimu However, this provides better performance than FIFO.

For example, the requests for I/O to blocks are on the following cylinders

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

and current head position is 143 cylinder then, it will first service to the reque for cylinder 130 not for cylinder 86, since 130 is closest to 143.

So the servicing order will be as follows –

130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774

This can be shown by fig. 2.19 as follows –

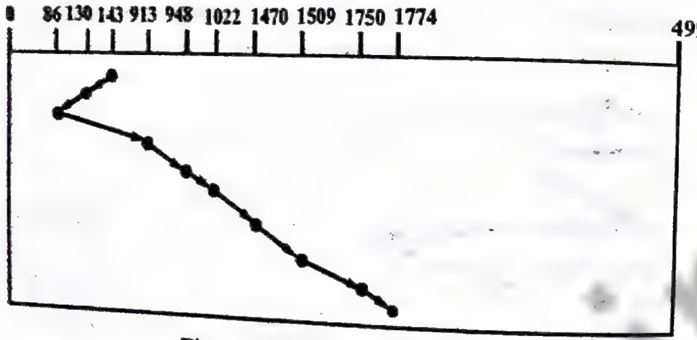


Fig. 2.19 SSTF Scheduling

The total head movements

$= 13 + 44 + 827 + 35 + 74 + 448 + 39 + 241 + 24 = 1745$ cylinders

This scheduling requires total 1745 head movements which is almost one fifth of FCFS requirement. So it is more improved form of scheduling than FCFS. The problem with this scheduling is that it may cause ***starvation*** of some requests.

Q.44. Write short note on SCAN type disk scheduling.

Ans. In SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder. until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. Thus, the head continuously scans back and forth across the disk.

It is necessary to know the direction of head movement, in addition to the head's current position. SCAN operates like SSTF except that it chooses the request that result in the shortest seek distance in a preferred direction.

If the preferred direction is currently outward, then the SCAN strategy chooses the shortest seek distance in the outward direction. SCAN does not change direction until it reaches the outermost cylinder or until there are no further requests pending in the preferred direction. In this sense, it is sometimes called the ***elevator algorithm*** because an elevator normally continues in one direction until there are no more requests pending and then it reverses direction.

Let us take an example, where requests for the blocks on cylinder given as 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 and current location of head is at 143 cylinder as well as the disk arm is moving toward 0.

Now, by using SCAN scheduling algorithm, it first serves for cylinder 130, which is first request in direction of minimum limit of disk. When it reaches 0, it reverses its direction and starts servicing in that direction.

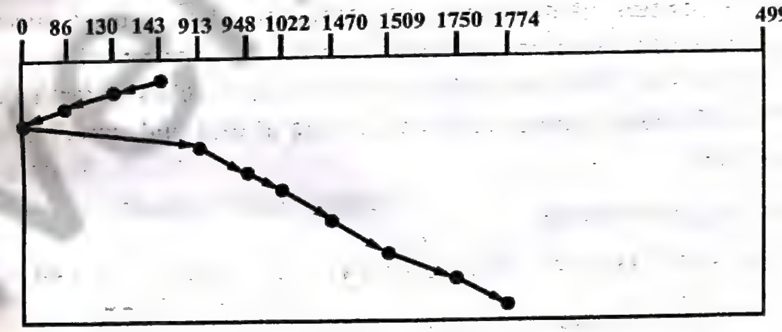


Fig. 2.20 SCAN Scheduling

The total head movements

$= 13 + 44 + 86 + 913 + 35 + 74 + 448 + 39 + 241 + 24$

$= 1917$ cylinders

So, total head movement is slightly larger than head movements for SSTF.

Q.45. Explain circular-scan (C-SCAN) type disk scheduling.

Ans. Circular SCAN (C-SCAN) scheduling is a variant of SCAN scheduling. It works just like SCAN scheduling, by moving head in a direction and satisfying requests along the way. When head reaches at the other end, it immediately returns to the beginning of disk without servicing in reverse order. Thus, C-SCAN completely eliminates the discrimination against requests for the innermost or outermost cylinders. It has a very small variance in response times.

Let us take same example where requests for cylinder are as follows

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

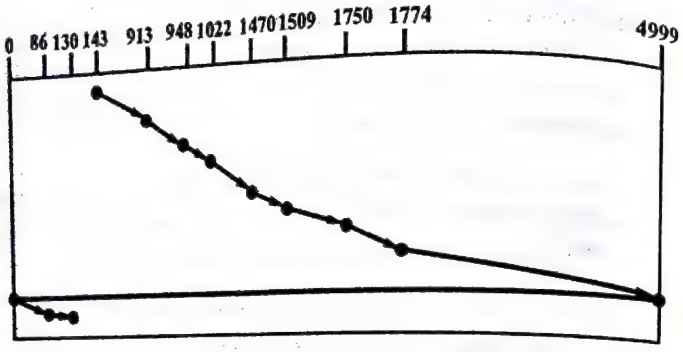


Fig. 2.21 C-SCAN Scheduling

and current position of head is at 143 cylinder. Then if head moves i maximum limit direction, it will first serve for cylinder 913 then to 94 1022, 1470, 1509, 1750 and at last to 1774 then reaches at last of dis i.e., 4999. After that it jumps to the beginning of disk and starts servici to 86 and 130.

Total head movements

= 770 + 35 + 74 + 448 + 39 + 241 + 24 + 3225 + 86 + 44

= 4986 cylinders

Q.46. Compare the throughput of C-SCAN and SCAN, assuming uniform distribution of requests. ***(R.G.P.V., June 200***

Ans. Assuming a uniform distribution of requests for cylinders, conside the density of requests when the head reaches one end and reverses directio At this point, relatively few requests are immediately in front of the head since these cylinders have recently been serviced. The heaviest density o requests is at the other end of the disk. These requests have also waited th longest, so we must go there first. This is the idea C-SCAN algorithm.

Q.47. Describe about Look type disk scheduling.

Ans. If SCAN and C-SCAN scheduling are implemented in such a wa that the disk arm goes only as far as the final request in each direction i satisfied, then it reverses direction immediately, without going all the way t the end of the disk. These versions of SCAN and C-SCAN are known a LOOK and C-LOOK scheduling.

Fig. 2.22 shows the implementation of C-LOOK algorithm.

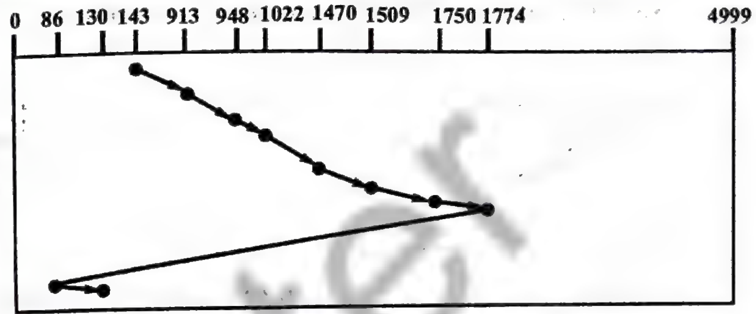


Fig. 2.22 C-LOOK Scheduling

Total head movements

= 770 + 35 + 74 + 448 + 39 + 241 + 24 + 44

= 1675 cylinders

Q.48. Compare the merits and demerits of various disk scheduling algorithms. ***(R.G.P.V., June 2016)***

Ans. (i) FCFS –

Merits – Some merits are –

(a) Every request gets a fair chance.

(b) No indefinite postponement.

(c) No reordering of work queue.

(d) Performs operations in order requested.

Demerits – Some demerits are –

(a) Does not try to optimize seek time.

(b) May not provide the best possible service.

(c) Poor performance.

(ii) SSTF –

Merits – Some merits are –

(a) Reduces total seek time compared to FCFS.

(b) Average response time decreases.

(c) Throughput increases.

Demerits – Some demerits are –

(a) Overhead to calculate seek time in advance.

(b) Switching directions slows things down.

(c) Starvation is possible.

(iii) SCAN –

Merits – Some merits are –

(a) Reduces variance compared to SSTF.

(b) High throughput.

Demerit – The main demerit is long waiting time for reque for locations just visited by disk arm.

(iv) C-SCAN –

Merits – Some merits are –

(a) Provide more uniform wait time compared to SCAN.

(b) Completely eliminates the discrimination against request the innermost or outermost cylinders.

(c) It has a very small variance in response times.

(v) LOOK and C-LOOK –

Merit – The main merit is that it prevents the extra delay whi occurred due to unnecessary traversal to the end of the disk.

Q.49. Discuss FCFS scheduling with example. Also discuss t advantages of FCFS. **(R.G.P.V., June 201**

Ans. Refer Q.42 and Q.48 (i).

NUMERICAL PROBLEMS

Prob.2. Suppose that a disk drive has 5000 cylinders, numbered 0 t 4999. The drive is currently serving a request at cylinder 143 and the previo request was at cylinder 125. The queue of pending requests, in FIFO order, is

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

Starting from the current head position, what is the total head distanc (in cylinders) that the disk arm moves to satisfy all pending requests fo each of the following disk scheduling algorithms.

FCFS, SSTF, SCAN, LOOK, C-SCAN, C-LOOK.

(R.G.P.V., June 200

Or

Suppose that a disk has 5000 cylinders. The drive is currently serving request at cylinder 143 and the previous request was at cylinder 125. Th queue of pending request in FIFO orders is 86, 1470, 913, 1774, 948, 150 1022, 1750 and 130. What is the total distance that the disk arm moves fo the following algorithms ?

(i) FCFS (ii) SSTF (iii) LOOK (iv) C-SCAN.

(R.G.P.V., Dec. 201

Or

The queue of pending requests, in FIFO order, is –

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.

Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling –

(i) FCFS (ii)SSTF (iii) SCAN (iv) LOOK (v) C-SCAN.

(R.G.P.V., Dec. 2012)

Or

Suppose that a disk drive has 5000 cylinders numbered from 0 to 4999. The drive is currently serving a request at cylinder 143 and previous request was at cylinder 125. The queue of pending request in FIFO order is – 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 starting from the current head position. What is the total distance (in cylinder) that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms –

(i) FCFS (ii) LOOK.

(R.G.P.V., Dec. 2013)

Sol. *(i)* ***FCFS*** – Refer Q.42.

(ii) ***SSTF*** – Refer Q.43.

(iii) ***SCAN*** – Refer Q.44.

(iv) ***C-SCAN*** – Refer Q.45.

(v) ***Look*** – Refer Q.47.

Prob.3. Suppose a disk drive has 300 cylinders, numbered 0 to 299. The current position of the drive is 90. The queue of pending requests in FIFO order is –

36, 79, 15, 120, 199, 270, 89, 170.

Calculate the average cylinder movements for SSTF algorithm.

(R.G.P.V., June 2010)

Sol. In this, the head first moves from 90 to 89 and 79, 120, 170, 199, 270, 36 and finally 15. This schedule is shown in fig. 2.23.

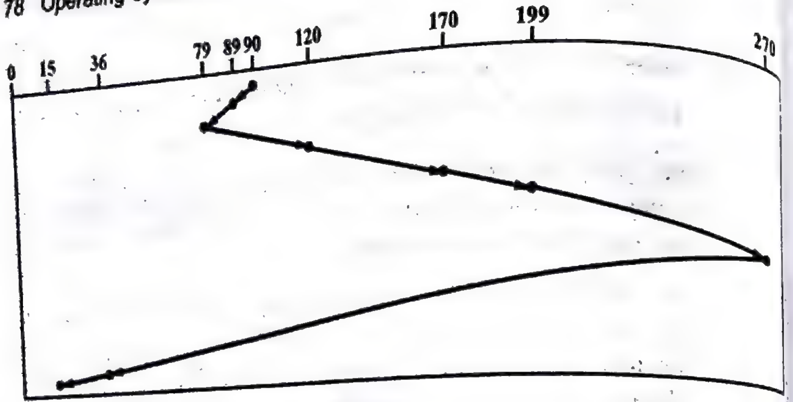


Fig. 2.23

Total head movement

$$= (90-89)+(89-79)+(120-79)+(170-120)+(199-170)+(270-199)+(270-36)+(36-15)$$

$$= 1+10+41+50+29+71+234+21 = 457 \text{ cylinders.}$$

Prob.4. Suppose that head of a moving head disk with 200 tracks numbered 0 to 199 is currently serving a request at 50 track and has just finished a request at track 85. If the queue of requests is kept in FIFO order 100, 199, 56, 150, 25, 155, 70 and 85. A seek takes 6m sec per cylinder moved. How much seek time is needed for the SCAN ?

(R.G.P.V., Dec. 2014)

Sol. The scheduling is shown in fig. 224.

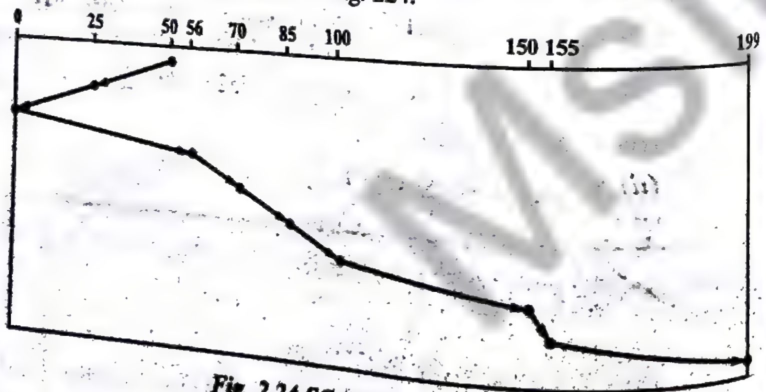


Fig. 2.24 SCAN Scheduling

Total head movement = 25 + 25 + 56 + 14 + 15 + 15 + 150 + 15 + 44.

There are total 249 cylinders moves required to complete the request queue.

Total seek time = 6 × 249 = 1494 ms = 1.494 sec

Prob.5. Suppose that a disk has 4000 cylinders numbered 0 to 3999. The drive is currently serving a request at cylinder 143 and the previous request was at 125. The queue of pending request in FIFO order is 86, 1470, 913, 1774, 948, 1500, 1020, 1751, 132.

Starting from the current head position, what is the total difference that the disk arm moved to satisfy all the pending request for each of the following disk scheduling algorithm –

(i) FCFS (ii) SCAN (iii) C-SCAN (iv) SSTF (v) C-Look (vi) Look.

(R.G.P.V., Dec. 2010)

Sol. (i) FCFS – Initially the disk head is at cylinder 143. Using FCFS scheme, the arm will go next to cylinders 86, 1470, 913, 1774, 948, 1500, 1020, 1751, and finally to 132. This schedule is shown in fig. 2.25.

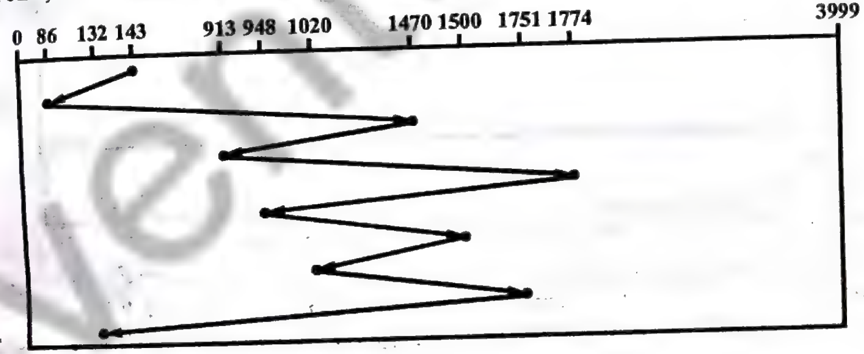


Fig. 2.25 FCFS Scheduling

Total head movement = (143 – 86) + (1470 – 86) + (1470 – 913) + (1774 – 913) + (1774 – 948) + (1500 – 948) + (1500 – 1020) + (1751 – 1020) + (1751 – 132)

= 57 + 1384 + 557 + 861 + 826 + 552 + 480 + 731 + 1619

= 7067 cylinders

(ii) SCAN – This scheduling is shown in fig. 2.26.

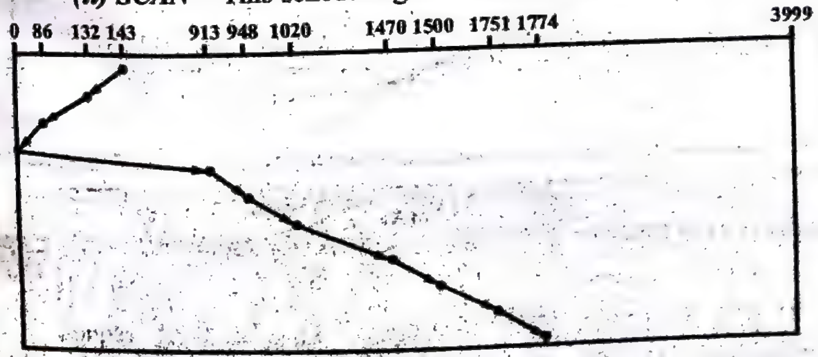


Fig. 2.26 SCAN Scheduling

Total head movement = (143 – 132) + (132 – 86) + (86 – 0) + (913 – 0)
+ (948 – 913) + (1020 – 948) + (1470 – 1020)
+ (1500 – 1470) + (1751 – 1500) + (1774 – 1751)
= 11 + 46 + 86 + 913 + 35 + 72 + 450 + 30 + 251 + 23
= 1917 cylinders

(iii) C-SCAN – This scheduling is shown in fig. 2.27.

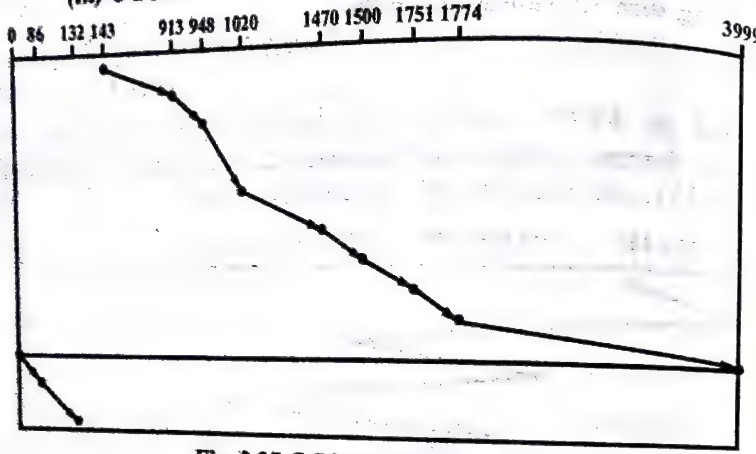


Fig. 2.27 C-SCAN Scheduling

Total head movement = (913 – 143) + (948 – 913) + (1020 – 948) + (1470
– 1020) + (1500 – 1470) + (1751 – 1500) + (1774
– 1751) + (3999 – 1774) + (86 – 0) + (132 – 86)
= 770 + 35 + 72 + 450 + 30 + 251 + 23 + 2225 + 86 + 46
= 3988 cylinders

(iv) SSTF – This scheduling is shown in fig. 2.28.

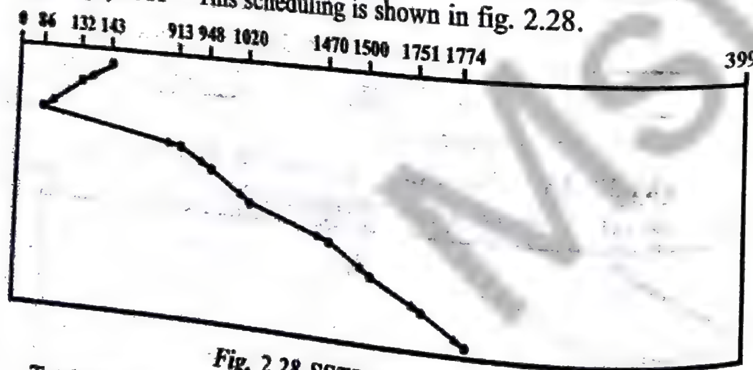


Fig. 2.28 SSTF Scheduling

Total head movement = (143 – 132) + (132 – 86) + (913 – 86) + (948
– 913) + (1020 – 948) + (1470 – 1020) + (1500
– 1470) + (1751 – 1500) + (1774 – 1751)
= 11 + 46 + 827 + 35 + 72 + 450 + 30 + 251 + 23
= 1745 cylinders

(v) C-Look – This scheduling algorithm is shown in fig. 2.29.

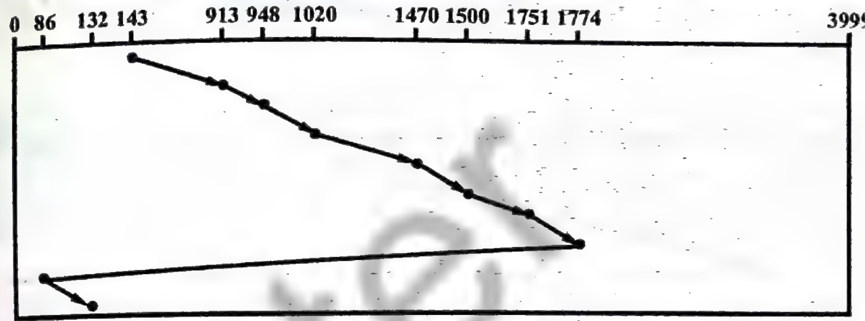


Fig. 2.29 C-Look Scheduling

Total head movement = (913– 143) + (948– 913) + (1020 – 948) + (1470
– 1020) + (1500 – 1470) + (1751 – 1500) + (1774
– 1751) + (132 – 86)
= 770 + 35 + 72 + 450 + 30 + 251 + 23 + 46
= 1677 cylinders

(vi) Look – This scheduling algorithm is shown in fig. 2.30.

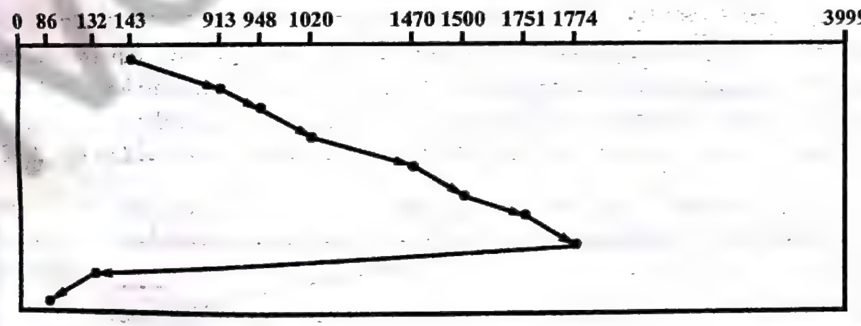


Fig. 2.30 Look Scheduling

Total head movement = (913 – 143) + (948 – 913) + (1020 – 948)
+ (1470 – 1020) + (1500 – 1470) + (1751 – 1500)
+ (1774 – 1751) + (1774 – 132) + (132 – 86)
= 770 + 35 + 72 + 450 + 30 + 251 + 23 + 1642 + 46
= 3319 cylinders

Prob.6. The request tracks in the order received are 55, 58, 39, 18, 90, 160, 150, 38, 98. Apply the following disk scheduling algorithms starting tracks at 100.

(i) SSTF
(ii) C-SCAN.

(R.G.P.V., May 2018)

Sol. (i) SSTF – According to the given problem the head is on 100 and it will more towards 98, 90, 58, 55, 39, 38, 18, 150 and finally 160. This schedule is shown in fig. 2.31.

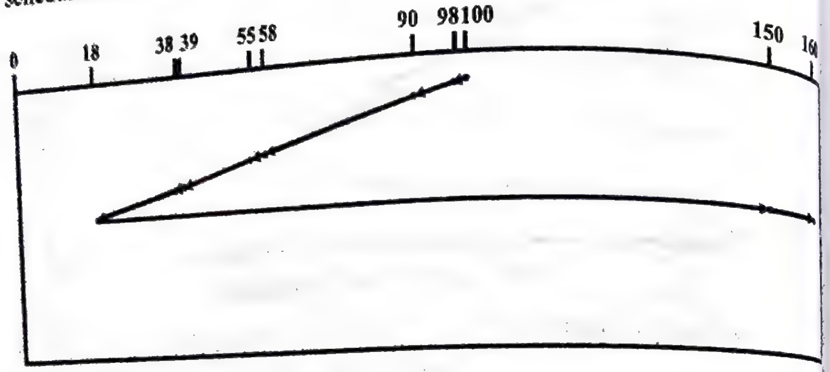


Fig. 2.31

∴ The total head movements

= 2 + 8 + 32 + 3 + 16 + 1 + 20 + 132 + 10

= 224 cylinders

(ii) C-SCAN – Consider the disk with 200 tracks numbered 0 to 199. Here the current position of the head is at 100 cylinder. Therefore the head moves in maximum limit direction, it will move forwards 150, 160 and then it will reach at the last of disk, the after it again jump to the beginning of disk and starts servicing to 18, 38, 39, 55, 58, 90 and 98 as shown in fig. 2.32.

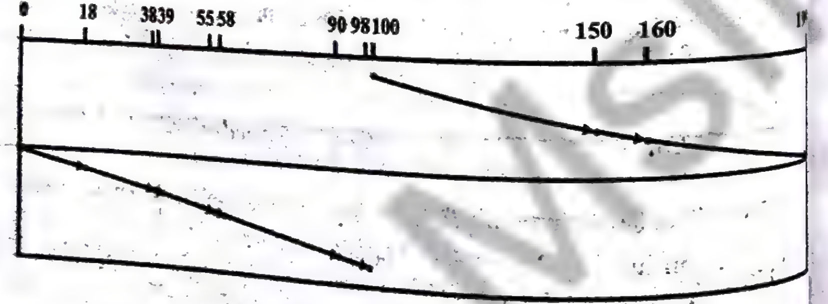


Fig. 2.32

Total head movements = 50 + 10 + 39 + 18 + 20 + 1 + 16 + 3 + 32 + 8

= 197 cylinders

⌘⌘

UNIT 3

CPU SCHEDULING – PROCESS CONCEPT, SCHEDULING CONCEPTS, TYPES OF SCHEDULERS, PROCESS STATE DIAGRAM, SCHEDULING ALGORITHMS

Q.1. Define program, process and processors. ***(R.G.P.V., June 2016)***

Ans. A process is a program in execution. It consists of the executable program, the program's data and stack, its program counter, stack pointer, and other registers, and all the other information needed to run the program. A program by itself is not a process. A program is a passive entity, such as the contents of a file stored on disk whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

Processor controls the operation of the computer and performs its data processing functions. When there is only one processor it is often referred to the CPU.

Q.2. What are the different operations performed on processes? Discuss in brief.

Ans. A system that manage processes must be able to perform certain operations on and with processes. These operations are –

(i) Create a process
(ii) Destroy a process
(iii) Suspend a process
(iv) Resume a process
(v) Change a process's priority
(vi) Block a process
(vii) Wakeup a process
(viii) Dispatch a process
(ix) Enable a process to communicate with another process, i.e., interprocess communication.

A process may create a new process through a create-process system call during the course of execution. The creating process is called the ***parent process*** and the created process is called the ***child process.*** Each of these new

processes may in turn create other processes forming a tree of processes as shown in fig. 3.1.

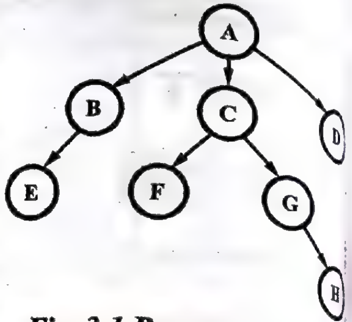


Fig. 3.1 Process Creation Hierarchy

Destroying a process involves removing it from the system. Its resources are returned to the system, it is purged from any system lists or tables, and its PCB is erased. Destruction of a process is complicated when the process has spawned other processes. In some systems, a spawned process is destroyed automatically when its parent is destroyed; in other systems, spawned processes proceed independently of their parents and the destruction of a parent has no effect on the destroyed parent's children.

A suspended process cannot proceed until another process resumes it. Suspensions last only brief periods of time. They are performed by the system to remove certain processes temporarily to reduce the system load during a peak loading situation. For long-term suspensions, the process's resources should be freed. Changing the priority of a process involves nothing more than modifying the priority value in the process's control block.

Q.3. Write short note on process control block (PCB).

(R.G.P.V., Dec. 2006, June 2010, May 2018)

Or

Write the use of process control block and discuss its contents.

(R.G.P.V., Dec. 2019)

Ans. Each process is represented in the operating system by a process control block (PCB) or a task control block. A PCB is shown in fig. 3.2.

PCB is used for storing the collection of information about the processes and this is also called as the data structure which stores the information about the process. The information of the process is used by the CPU at the run time. The various information which is stored into the PCB include the following –

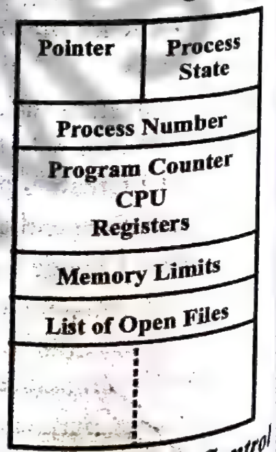


Fig. 3.2 Process Control Block (PCB)

(i) Process State – The state may be new, ready, running, waiting, halted and so on.

(ii) Program Counter – The counter specifies the address of the next instruction to be executed for this process.

(iii) CPU Registers – They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved to continue the process correctly afterward, when an interrupt occurs.

(iv) CPU-scheduling Information – It includes a process priority, pointers to scheduling queues and any other scheduling parameters.

(v) Memory Management Information – It includes the information such as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

(vi) Accounting Information – This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers and so on.

(vii) I/O Status Information – This information includes the list of I/O devices allocated to this process, a list of open files and so on.

Hence, the PCB serves as the repository for any information that may vary from process to process.

Q.4. List the attributes of PCB. ***(R.G.P.V., Dec. 2014)***

Ans. The attributes of PCB are –

(i) Process-id	(ii) Process state
(iii) Process priority	(iv) Register save area
(v) Pointers to the process memory	(vi) Pointers to other resources
(vii) List of open files	(viii) Accounting information
(ix) Other information	(x) Pointers to other PCBs.

Q.5. What is scheduling ? Why CPU scheduling is necessary ?

Ans. Scheduling refers to a set of policies and mechanism built into the operating system that govern the order in which the work to be done by a computer system is completed. A schedular is an OS module that selects the next job to be admitted into the system and the next process to run.

CPU scheduling is the basis of multiprogrammed OS by switching the CPU among processes, the operating system can make the computer more productive.

The objective of multiprogramming is to have some process running at all times in order to maximize CPU utilization. In uniprocessor system, only one process may run at a time, any other processes must wait until the CPU is free and can be rescheduled. With multiprogramming several processes are kept in memory at one time. When one process has to wait, the operating system takes the CPU away from that process and give the CPU to another process.

Scheduling is a fundamental operating-system function. Almost all computer resources are scheduled before use. The CPU is one of the primary computer resources. Thus, its scheduling is central of operating system design.

There are number of situation where CPU scheduling is needed –

(i) When a new process is created.

(ii) When a process terminates.

(iii) When a process wait for some I/O.

(iv) When an I/O interrupt occur.

Q.6. Write short note on I/O scheduler.

Ans. The I/O scheduler sequences the IORB chained to a DCB according to the scheduling policy and picks up an IORB from that chain when a request is to be serviced. It then requests the device handler to carry out the I/O operation and then deletes that IORB from all the queues for pending requests from the appropriate DCB, CUCB and CCB after setting the appropriate flags in them to denote that they are now busy. The I/O scheduler uses a number of policies for scheduling IORBs. In fact, IORBs are chained to one another and to the DCB depending upon this policy only. For example, if first come first served (FCFS) method is used, the new IORB is added at the end of the queue. Therefore, it can be said that the I/O procedure prepares the IORB and hands it over to the I/O scheduler. The I/O scheduler then chains it with the DCB as per its scheduling policy. In modern operating systems, scheduling is done by the controller hardware itself thereby making the task of the I/O scheduler easier and the entire operation much faster. In some others, the software in operating system has to carry it out.

Q.7. Explain about various types of schedulers you know. Also discuss the purpose of each.
(R.G.P.V., Dec. 2005)

Or

Explain short-term, medium-term and long-term scheduling.
(R.G.P.V., Dec. 2011)

Or

What are different process scheduling levels ? How do they interact with each other ?
(R.G.P.V., Dec. 2013)

Or

Describe the differences among short term, medium term and long term scheduling.
(R.G.P.V., Dec. 2016)

Or

Explain the following terms –

(i) Long-term scheduler

(ii) Short-term schedule

(iii) Medium-term scheduler.
(R.G.P.V., Dec. 2008)

Ans. There are three types of schedulers – *long-term, medium-term* and *short-term* schedulers.

Fig. 3.3 shows the possible traversal paths of jobs and programs through the components and queues, depicted by rectangles, of a computer system. The primary places of action of the three types of schedulers are marked with down arrows. In fig. 3.3, a submitted batch job joins the batch queue while waiting to be processed by the long-term scheduler. Once scheduled for execution, processes spawned by the batch job enter the ready queue to await processor allocation by the short-term scheduler. After becoming suspended, the running process may be removed from memory and swapped out to secondary storage. Such processes are subsequently admitted to main memory by the medium-term scheduler for execution by the short-term scheduler.

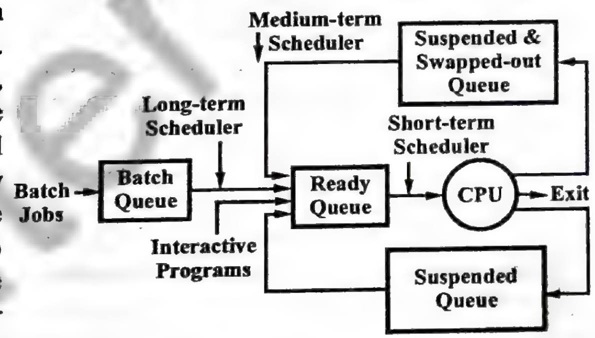


Fig. 3.3 Schedulers

(i) The Long-term Sheduler or Job Scheduler – In batch system, more processes are submitted than can be executed immediately. These processes are spooled to a mass-storage device such as a disk, where they are kept for later execution. The long-term scheduler selects processes from this pool and loads them into memory for execution.

(ii) The Medium-term Scheduler – After executing for a while, a running process may become suspended by making an I/O request or by issuing a system call. It is sometimes beneficial to remove these suspended processes from main memory to make room for other processes. In practice, the main-memory capacity may impose a limit on the number of active processes in the system. When a number of those processes become suspended, the remaining supply of ready processes in systems where all suspended processes remains resident in memory may become reduced to a level that weakens functioning of the short-term scheduler by leaving it few or no options for selections. This problem may be alleviated by moving suspended processes to secondary storage, in systems with no support for virtual memory. Thus, saving the image of a suspended process in secondary store is called swapping and the process is said to be swapped out or rolled out.

The medium-term scheduler is incharge of handling the swapped-out processes. Once the suspending condition is removed, the medium-term scheduler attempts to allocate the required amount of main memory, and swap the process in and make it ready. The medium-term scheduler must be provided with information about the memory requirements of swapped-out processes to work properly.

(iii) The Short-term Sheduler or CPU Scheduler – The short-term scheduler selects from among the pool of processes resident in memory that are ready to execute, and allocates the CPU to one of them.

Q.8. What is context switching ? Discuss different types of scheduler.
(R.G.P.V., June 2017)

***Ans.* Context Switching** – Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch. The context of a process is represented in the PCB (process control block) of a process; it includes the value of the CPU registers, the process state and memory management information. When a context switch occurs, the kernel saves the context of the old process in it PCB and loads the saved context of the new process scheduled to run. Context switch time is pure overhead, because the system does not useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied and the existence of special instructions. Typical speeds range from 1 to 1000 microseconds. Context switch times are highly dependent on hardware support.

Different Types of Scheduler – Refer Q.7.

Q.9. Define process states. Draw the diagram of PCB.
(R.G.P.V., June 2017)

***Ans.* Process State** – A process goes through a series of discrete process states. Various events can cause a process to change states. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states –

(i) ***New*** – The process is being created.

(ii) ***Running*** – Instructions are being executed.

(iii) ***Waiting*** – The process is waiting for some event to occur such as an I/O completion or reception of a signal.

(iv) ***Ready*** – The process is waiting to be assigned to a processor.

(v) ***Terminated*** – The process has finished execution.

A state diagram corresponding to these states is shown in fig. 3.4.

Diagram of PCB – Refer fig. 3.2, Q.3.

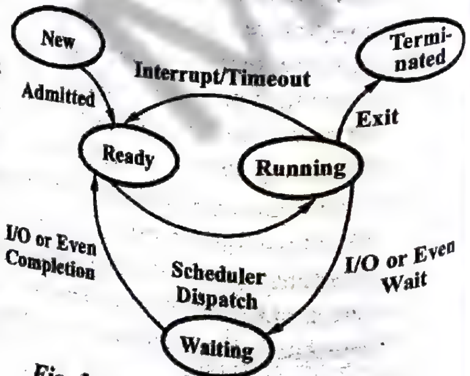


Fig. 3.4 Diagram of Process State

Q.10. Draw and explain process state transition diagram with the following states –

(i) New	***(ii) Suspend ready***	***(iii) Ready***
(iv) Running	***(v) Exit***	***(vi) Blocked***
(vii) Suspend block.		

(R.G.P.V., June 2006)

Ans. When a job is admitted to the system, a corresponding process is created and inserted at the back of the ready list. When the process reaches the head of the list, and when the CPU becomes available, the process is given the CPU and is said to make a state transition from ready state to the running state as shown in fig. 3.5. The assignment of the CPU to the first process on the ready list is called ***dispatching,*** and is performed by the ***dispatcher.*** This transition is indicated as follows –

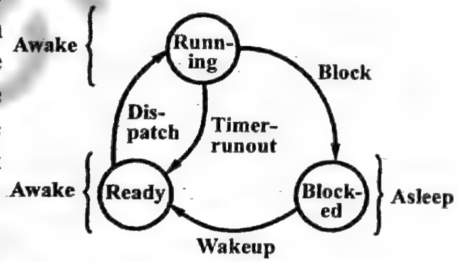


Fig. 3.5 Process State Transitions

dispatch (processname) : ready → running

To prevent any process from monopolizing the system, the operating system makes the previously running process ready, and makes the first process on the ready list running. These state transitions are indicated as –

timerrunout (processname) : running → ready

and dispatch (processname) : ready → running

If a running process initiates an I/O operation before its quantum expires, the running process voluntarily relinquishes the CPU, i.e., the process blocks itself pending the completion of the I/O operation. This state transition is

block (processname) : running → blocked

The process makes the transition from the blocked state to the ready state. The transition is

wakeup (processname) : blocked → ready

Fig. 3.6 shows the process state transition diagram modified to include suspend and resume. Two new states have been added, namely suspendedready and suspendedblocked. In fig. 3.6, active states are above the dashed line and below it are suspended states.

A suspension may be initiated either by a process itself or by another process. On a uniprocessor system a running process may suspend itself. On a multiprocessor system, a running process may be suspended by another process running at that moment on a different processor.

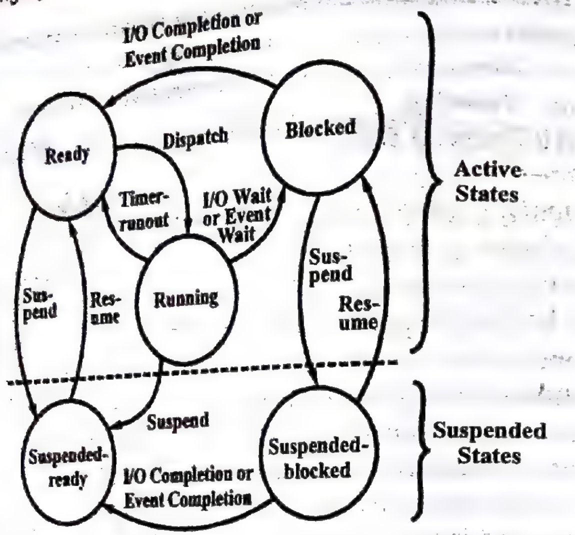


Fig. 3.6 Process State Transition with Suspend and Resume

A ready process may be suspended only by another process. It makes the transition

suspend (processname) : ready $\rightarrow$ suspendedready

A suspendedready process may be made ready by another process. It makes the transition

resume (processname) suspendedready $\rightarrow$ ready

A blocked process may be suspended by another process. It makes the transition.

suspend (processname) : blocked $\rightarrow$ suspendedblocked

A suspended blocked process may be resumed by another process. It makes the transition.

resume (processname) : suspended blocked $\rightarrow$ blocked

Q.11. What are the various types of scheduling algorithms ? Discuss any one.

Ans. Following are the various types of scheduling alogorithms –

(i) First-come-first-serve (FCFS) CPU scheduling
(ii) Shortest-job-first (SJF) CPU scheduling
(iii) Shortest-remaing-time-first (SRTF) CPU scheduling
(iv) Priority CPU scheduling
(v) Round robin CPU scheduling
(vi) Multilevel queue scheduling
(vii) Multilevel feedback queue scheduling.

First-come-first-serve (FCFS) CPU Scheduling – This is simplest CPU scheduling algorithm . The idea is that the job which comes first in ready queue is executed first by the CPU.

In this method, ready queue is considered as a FIFO queue, i.e., the job comes in ready queue is added at the tail of the queue and the CPU takes the process from the head of the queue for execution.

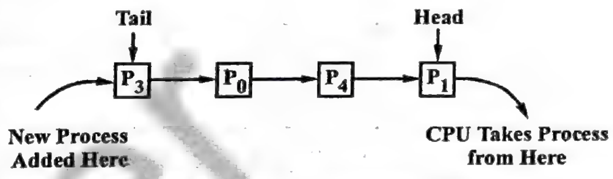


Fig. 3.7 Ready Queue in FCFS

Example –

Process	*CPU Burst Time*
P_1	20 ms
P_2	4 ms
P_3	5 ms

Let the process P_1, P_2 and P_3 arrives at the same time 0. The CPU burst time also shown here. Suppose processes arrive in order P_3, P_2, P_1 and served in FCFS order then Gantt chart is as follows –

P_3	P_2	P_1
0 – 5	5 – 9	9 – 29

Waiting time for process $P_1 = 9$ ms

Waiting time for process $P_2 = 5$ ms

Waiting time for process $P_3 = 0$ ms

Total waiting time = 14 ms

So, average waiting time = 14/3 = 4.67 ms

This average waiting time depends upon the order of the arriving.

This policy is non-preemptive, i.e., once the CPU has been allocated to a process, it keeps the CPU until it releases the CPU either by terminating or by requesting I/O.

Q.12. Explain about shortest-job-first (SJF) CPU scheduling.

Ans. SJF is very good CPU scheduling. In this, the job that has the smallest next CPU burst, is allocated to the CPU for executing first. In case, where more than one jobs have same length of next CPU burst, the FCFS scheduling is used to break the tie. This scheduling should be known as ***shortest next CPU burst*** rather than shortest job first.

Example –

Process	CPU Burst Time
P_1	7
P_2	8
P_3	5
P_4	4

Let all jobs P_1, P_2, P_3 and P_4 comes at the same time. Then Gantt chart according to SJF is as follows –

P_4	P_3	P_1	P_2
0 – 4	4 – 9	9 – 16	16 – 24

Waiting time for process P_1 = 9 ms
Waiting time for process P_2 = 16 ms
Waiting time for process P_3 = 4 ms
Waiting time for process P_4 = 0 ms
Total waiting time = 29 ms
Average waiting time = 29/4 = 7.25 ms

The main advantage of this policy is that, it is optimal scheduling in connection with waiting time.

The disadvantage of this is that, it cannot be implemented for the short-term scheduler, since there is no way to find the length of the next CPU burst.

Q.13. Describe shortest-remaining-time-first (SRTF) CPU scheduling.

Ans. The preemptive version of SJF scheduling is known as shortest-remaining-time-first scheduling. The idea is that, if a new process arrives in memory and its next CPU burst is shorter than remaining CPU burst of currently running process, then currently running process is put into the ready queue and CPU is allocated for the new process.

Example –

Process	Arrival Time	CPU Burst Time
P_1	0	8
P_2	2	5
P_3	3	9
P_4	4	3

For the processes P_1, P_2, P_3, P_4 arrival time and CPU burst time is shown above. The Gantt chart according to SRTF is as follows –

P_1	P_2	P_4	P_1	P_3
0 – 2	2 – 7	7 – 10	10 – 16	16 – 25

Explanation – Process P_1 arrives at time 0, so it first executed. When process P_1 is under execution at time 2, process P_2 arrives. The time remaining for process P_1 is (8 – 2 = 6) ms, which is larger than the time required by process P_2 (5 ms), so process P_1 is preempted and process P_2 is taken for execution. Similarly, we can get the waiting time for all processes.

Waiting time for process P_1 = 10 – 2 = 8 ms
Waiting time for process P_2 = 2 – 2 = 0 ms
Waiting time for process P_3 = 16 – 3 = 13 ms
Waiting time for process P_4 = 7 – 4 = 3 ms
Total waiting time = 24 ms
Average waiting time = 24/4 = 6 ms

If we use a non-preemptive SJF for this example, then average waiting time will be 6.5 ms.

Q.14. Explain about priority CPU scheduling.

Ans. The idea of priority-based scheduling is as follows –

A priority is given to a process on some basis, the CPU is allocated first to the highest priority process. If processes have equal priority then FCFS is used. Shortest job first scheduling can be thought as a priority algorithm.

If priority is assigned to each process on the basis of CPU burst like –

$$\text{Priority} \propto \frac{1}{\text{Length next CPU burst}}$$

Then the process with larger CPU burst has lower priority and vice versa.

The numbers are used to represent the priority.We will use low number to represent higher priority. The priority scheduling may either be preemptive or non-preemptive. In non-preemptive priority scheduling the new process is just put at the head of the ready queue, if its priority is highest than all processes in the ready queue and higher than the currently running process's priority. In preemptive priority scheduling, CPU preempts the currently running process, if its priority is lower than the priority of newly arrived process.

Example –

Process	CPU Burst Time	Priority
P_1	9	3
P_2	1	1
P_3	1	4
P_4	2	2

Here P_1, P_2, P_3, P_4 processes are shown with their priorities and CPU burst time. The P_2 has higher priority among P_1, P_3 and P_4.

The Gantt chart is

P2	P4	P1	P3
0 – 1	1 – 3	3 – 12	12 – 13

$$\text{Average waiting time} = \frac{0+1+3+12}{4} = 4 \text{ ms.}$$

Q.15. Describe round robin (RR) CPU scheduling.

Or

Discuss round robin scheduling policy with its merits and demerits. What is the impact of the quantum of time slice on the system performance?
(R.G.P.V., Dec. 2013)

Ans. The idea of RR scheduling is just like FCFS but it is preemptive. Time slice, is a small unit of time. The CPU is allocated to every process in the ready queue for one time slice. Here, ready queue is treated as a circular queue. The time slice may be of size 10 to 100 milliseconds. Due to this reason, it is called oftenly *time slice scheduling.*

The procedure of RR scheduling is as follows – new process is added to the tail of the queue. The process is picked for the execution from the head of the queue. The timer is set to interrupt after one time slice.

There may be cases as follows –

(i) In first case, process CPU burst time is less than 1 time slice, then it will be completely executed and will release CPU and next process is executed.

(ii) In second case, an interrupt occurs at 1 time slice. A context switch will be made, and process will be put at the tail of the ready queue. The next process is executed.

Example –

Process	*CPU Burst Time*
P_1	20
P_2	5
P_3	4

Let the size of time slice be 4 ms, P_1 takes CPU for first 4 ms. After that an interrupt occurs and P_1 is put into the tail of ready queue. P_2 is executed for 4 ms then it is put to the tail of ready queue. After that P_3 is executed for 4 ms, it is completed and releases the CPU without interrupt. Again the turn of P_1 comes for 4 ms. This process continues.

The Gantt chart –

P1	P2	P3	P1	P2	P1	P1	P1
0 – 4	4 – 8	8 – 12	12 – 16	16 – 17	17 – 21	21 – 25	25 – 29

Waiting time for process $P_1 = 0 + 8 + 1 = 9$ ms
Waiting time for process $P_2 = 4 + 8 = 12$ ms
Waiting time for process $P_3 = 8$ ms
Total waiting time $= 29$ ms
Average waiting time $= 29/3 = 9.67$

The performance of RR scheduling totally depends on the size of time slice. If time slice is too large, then RR is just like FCFS. If time slice is too small, then RR is just like process sharing, i.e., a process will take much time to wait.

Advantages – (i) It supports time sharing system.

(ii) It uses quantum.

(iii) It can handle multiple processes.

(iv) If process burst time is smaller than quantum, process excute in first quantum.

Disadvantages – (i) Process excution is slower if process burst time is large than quantum.

(ii) If quantum is large it works as FCFS.

(iii) Large process has to wait for ending of small process.

Q.16. Write about FCFS scheduling and round robin scheduling which one is best in which condition ? Justify your answer. (R.G.P.V., June 2017)

Ans. **FCFS CPU Scheduling** – Refer Q.11.

Round Robin Scheduling – Refer Q.15.

Best Condition for Both – FCFS scheduling is best when priority of process is according to the sequence it comes to the processor (i.e. higher priority process is comming before the lower priority process). If higher priority process comes first to the processor than according to the FCFS scheduling it will shedule first. FCFS is also best for overall time complexity and CPU utilization because there is no overhead of selecting process for scheduling.

Round robin scheduling is best when there are equal quality of processes in the queue. Round robin is also best for real time or multiuser operating systems because it provides equal time quantum for each process, seems all the processes are executing parallely.

Q.17. Explain about multilevel queue scheduling.

Ans. The processes can be classified according to their response time. In multilevel queue scheduling, multiple queues are prepared, one for different category processes. These queues are in a hierarchical manner, the top queue has highest priority and bottom queue has lowest priority. Which process should be assigned to which queue depends on process type. The scheduling algorithm applied to each queue is different. Processes cannot move between queues.

We can classify processes in following manner –

(i) System processes (ii) Interactive processes

(iii) Batch processes (iv) Student processes.

In fig. 3.8, we can see the priority of each queue for the above mentioned processes.

Batch processes can only be run, if queue of interactive processes and queue of system processes are empty. Similarly no interactive process can run until the queue of system processes is empty. The reason is, interactive processes queue has priority over batch processes queue and system processes has priority over interactive processes queue.

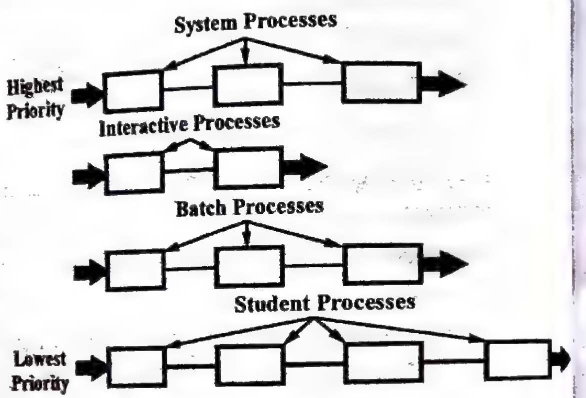


Fig. 3.8 Multilevel Queue Scheduling

Q.18. Describe multilevel feedback queue sheduling.

Ans. One of major disadvantages of multiple queue scheduling is indefinite postpondment of lower priority processes. To eliminate this problem, the multilevel feedback queue allows to move processes between queues.

Here idea is that processes are prioritized according to the CPU time, so that I/O bound processes found higher priority than CPU bound processes. So I/O bound processes are put in upper queues and CPU bound processes are put in lower queues. Process that wait to much time in lower queue may be moved to higher priority queue. This is called ***aging***.

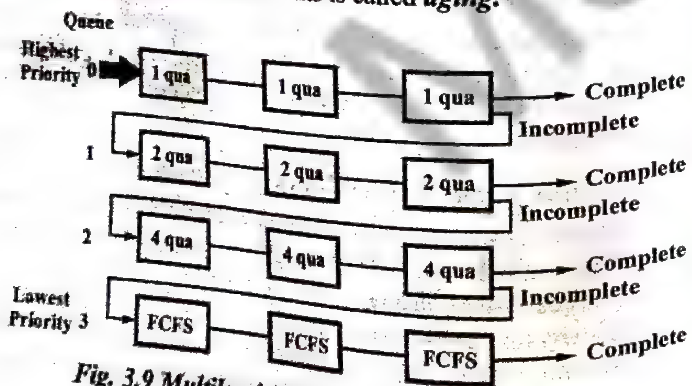


Fig. 3.9 Multilevel Feedback Queue Scheduling

We can see from the above fig. 3.9 that all the processes join the queuing system starting from queue 0. For each process in queue 0, one quantum CPU time is allocated. If process completes in one quantum time, process is thrown from the queuing system otherwise process put at the tail of queue 1. Similarly, for each process in queue 1, two quantum CPU time is allocated, if process completes in this time, process is thrown from the queuing system otherwise process put at the tail of queue 2. This continues until process reaches at lowest priority queue where processes are scheduled on FCFS basis.

Note that a process can be executed in any queue if all queues whose priorities are higher than this queue, are empty.

Q.19. What are preemptive and non-preemptive scheduling ? ***(R.G.P.V., Dec. 2012)***

Or

Explain preemptive scheduling. ***(R.G.P.V., Dec. 2013)***

Ans. A non-preemptive scheduling means that a running process preserves the control of the CPU and all the allocated resources until it surrenders control to the operating system on its own. This shows that even if a process of higher priority comes into the system, the running process should not give up the control. Whereas, if the running process becomes blocked due to any I/O request, another process is scheduled because, the waiting time for the I/O completion is increased. This scheduling is mostly used for achieving a higher throughput due to less overheads incurred in context switching, but it is generally not used for real time systems, where higher priority events require an immediate attention and thus, need to interrupt the recently running process.

A preemptive scheduling enables a higher priority process to change a recently running process even if its time slice is not finish or it has not requested for any I/O. This needs context switching repeatedly, thus decreasing the throughput, but then it is mostly used for on-line, real time processing, where interactive users and high priority processes need immediate attention.

Q.20. What is the difference between non-preemptive and preemptive scheduling ? Explain why it is not likely to use strict non-preemptive scheduling in a computer system ? ***(R.G.P.V., May 2018)***

Ans. Refer to Q.19.

Q.21. What are the main criteria used for comparing the CPU scheduling algorithm ?

Ans. Common performance measures and optimization criteria that schedulers may use in attempting to maximize system performance are –

(i) Processor Utilization – Processor utilization is the average fraction of time during which the processor is busy. Being busy refers to the

processor not being idle, and includes both time spent executing user program and executing the operating system. It is very necessary to keep the CPU as busy as possible, i.e., CPU should be utilized as much as possible. The scheduling algorithm that utilizes CPU very much, is considered better. CPU can be utilized from 0-100%.

(ii) Throughput – Throughput refers to the amount of work completed in a unit of time. One way to express throughput is by means of the number of user jobs executed in a unit of time. The higher the number, the more work is apparently being done by the system.

(iii) Turnaround Time – Turnaround time (T) is defined as the time that elapses from the moment a program or a job is submitted until it is completed by a system. It is the time spent in the system, and it may be expressed as a sum of the job service time (execution time) and waiting time.

(iv) Waiting Time – Waiting time (W) is the time that a process or a job spends waiting for resource allocation due to contentions with others in a multiprogramming system. Waiting time is the sum of the periods spent waiting in the ready queue. In other words, waiting time is the penalty imposed for sharing resources with others. Waiting time may be expressed as turnaround time less the actual execution time

$$W(x) = T(x) - x$$

where x is the service time, W(x) is the waiting time of the job requiring x units of service, and T(x) is the job's turnaround time.

(v) Response Time – A process can produce some output fairly early and can continue computing new results while results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure is called response time, i.e. the amount of time it takes to start responding, but not the time that it takes to output that response.

It is desirable to maximize CPU utilization and throughput, and to minimize turnaround time, waiting time, and response time.

Q.22. Explain scheduling queues with queuing diagram representation of process scheduling.

(R.G.P.V., June 200

Ans. As processes enter the system, they are put into a ***job queue***. The queue consists of all processes in the system. The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the ***ready queue***. This queue is generally stored as a linked list. A ready-queue header contains pointer to the first and final PCBs in the list. We extend each PCB to include a pointer field that points to the next PCB in the ready queue.

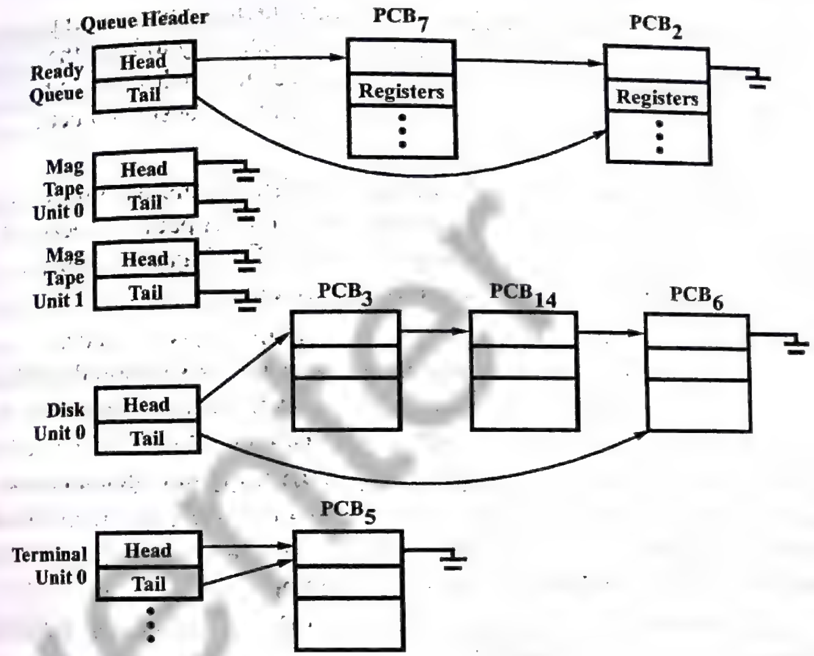


Fig. 3.10 Ready Queue and Various I/O Device Queues

The operating system also has other queues. When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request. In the case of an I/O request, such a request may be to a dedicated tape drive, or to a shared device, such as disk. Since the system has many processes, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk. The list of processes waiting for a particular I/O device is called a ***device queue***. Each device has its own device queue (fig. 3.10).

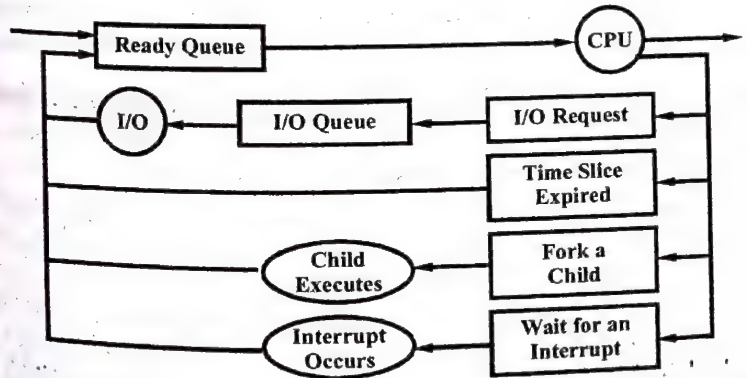


Fig. 3.11 Queuing-diagram Representation of Process Scheduling

The process scheduling can be represented by a queuing diagram as show in fig. 3.11. Each rectangular box represents a queue. Two types of queue are available – the ready queue and a set of device queues. The circles represen the resources that serve the queues, and the arrows indicate the flow of proces in the system.

A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution (or dispatched). Once the process i assigned to the CPU and is executing, one of several events could occur –

(i) The process could issue an I/O request, and then it will be placed in an I/O queue.

(ii) The process could create a new subprocess and wait for its terminatio

(iii) The process could be removed forcibly from the CPU, as result of an interrupt, and be put back in the ready queue.

In the first two cases, the process eventually switches from the waitin state to the ready state, and is then put back in the ready queue. A proces continues this cycle until it terminates, at which time it is removed from al queues and has its PCB and resource deallocated.

Q.23. Explain the differences in the degree to which the followin scheduling algorithms discriminate in favour of short processes –

(i) FCFS

(ii) Round robin

(iii) Multilevel feedback queues.

(R.G.P.V., June 2008)

Ans. (i) FCFS – It discriminates against short jobs since any short job arriving after long jobs will have a longer waiting time.

(ii) Round Robin – It treats all jobs equally, i.e., giving them equa bursts of CPU time, so short jobs will be able to leave the system faster sinc they will finish first.

(iii) Multilevel Feedback Queues – It discriminates very favourabl toward short jobs since they work similar to the round robin algorithm.

Q.24. Explain multilevel adaptive scheduling and fair-share schedulin with examples.

(R.G.P.V., Dec. 2010)

Ans. **Multilevel Adaptive Scheduling** – Refer Q.18.

Fair-share Scheduling – In a multiuser system, if individual use applications or jobs may be organized as multiple processes (or threads), the there is a structure to the collection of processes that is not recognized by traditional scheduler. From the user's point of view, the concern is not how particular process performs but rather how his or her set of processes, whic constitute a single application, performs. Thus, it would be attractive to mak

scheduling decisions on the basis of these process sets. This approach is generally known as ***fair-share scheduling***. The objective of a fair-share scheduler is to monitor usage to give less resources to users who have had more than their fair-share and more to those who have had less than their fair-share.

Fair-share scheduler (FSS) considers the execution history of a related group of processes, along with the individual execution history of each process in making scheduling decisions. The system divides the user community into a set of fair-share groups and allocates a fraction of the processor resource to each group. Thus, there might be four groups, each with 25% of the processor usage. In effect, each fair-share group is provided with a virtual system that runs proportionally slower than a full system.

Scheduling is done on the basis of priority, which takes into account the underlying priority of the process, its recent processor usage, and the recent processor usage of the group to which the process belongs. The higher the numerical value of the priority, the lower the priority. The following formulas apply for process j in group k –

$$CPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$GCPU_j(i) = \frac{CPU_j(i-1)}{2}$$

$$P_j(i) = Base_j + \frac{CPU_j(i-1)}{2} + \frac{GCPU_k(i-1)}{4 \times W_k}$$

where, $CPU_j(i)$ = Measure of processor utilization by process j through interval i

$GCPU_k(i)$ = Measure of processor ultilization of group k through interval i

$P_j(i)$ = Priority of process j at beginning of interval i; lower values equal higher priorities

$Base_j$ = Base priority of process j

W_k = Weighting assigned to group k, with the constraint that $0 \le W_k \le 1$ and $\sum_k W_k = 1$

Each process is assigned a base priority. The priority of a process drops as the process uses the processor and as the group to which the process belongs uses the processor. In the case of the group utilization, the average is normalized by dividing by the weight of that group. The greater the weight assigned to the group, the less its utilization will affect its priority.

Fig. 3.12 is an example in which process A is in one group and process B and process C are in a second group, with each group having a weighting of 0.5. Assume that all processes are processor bound and are usually ready to run. All processes have a base priority of 60. Processor utilization is measured as follows – The processor is interrupted 60 times per second; during each

interrupt, the processor usage field of the currently running process is incremented, as is the corresponding group processor field. Once per second, priorities are recalculated.

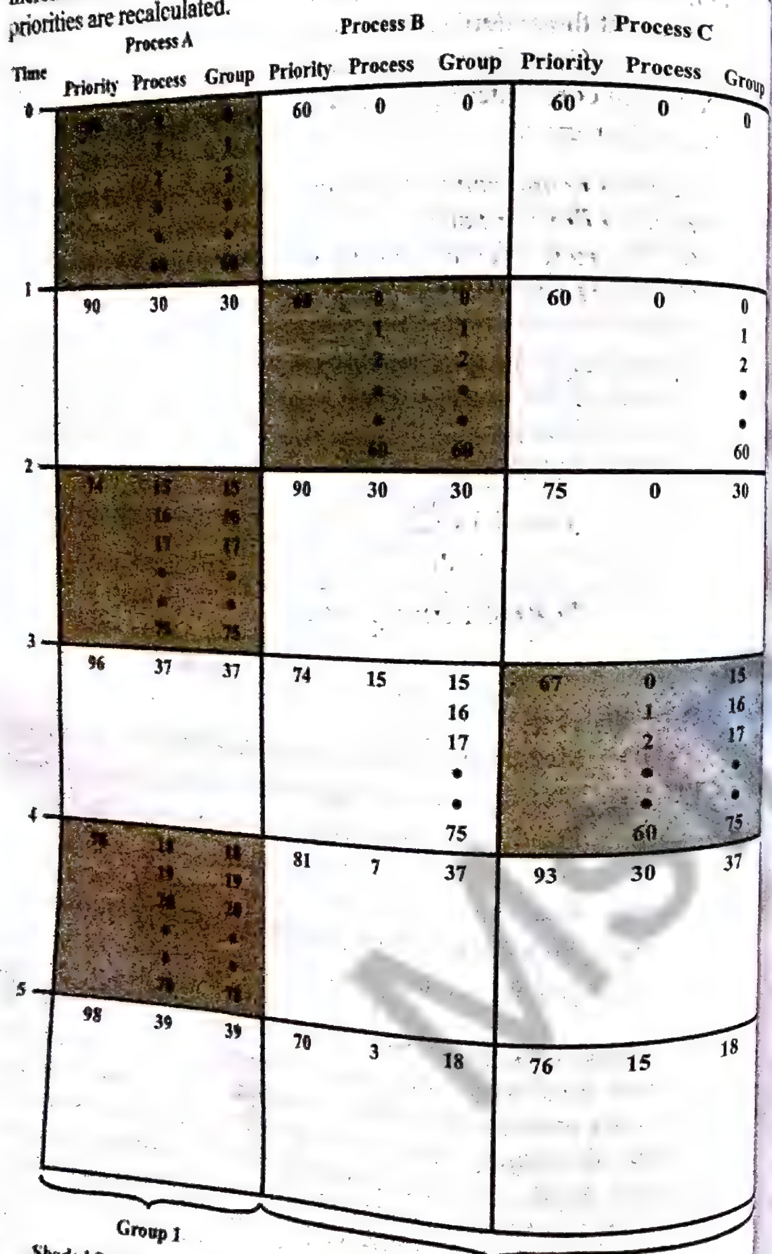


Fig. 3.12 Example of Fair-share Scheduler – Three Processes, Two Groups

In the figure, process A is scheduled first. At the end of one second, it is preempted. Processes B and C now have the higher priority and process B is scheduled. At the end of the second time unit, process A has the highest priority. Note that the pattern repeats – The kernel schedules the processes in order – A, B, A, C, A, B, etc. Thus 50% of the processor is allocated to process A, which constitutes one group and 50% to processes B and C, which constitute another group.

Q.25. Compare and contrast the highest-response ratio next scheduling policy with the following policies –

(i) Shortest time to go (STG) policy

(ii) Least completed next policy (iii) Round robin policy.

(R.G.P.V., Dec. 2010)

Ans. (i) Shortest Time to Go (STG) Policy – The shortest time to go (STG) policy reduces the bias in favour of long processes inherent in FCFS. This is a non-preemptive policy in which the process with the shortest expected processing time is selected next.

Fig. 3.13 and table 3.1 show the results for our example.

Table 3.1 Process Scheduling Example

Process	*Arrival Time*	*Service Time*
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

(ii) Least Completed Next Policy – The least completed next policy is a preemptive version of STG. In this case, the scheduler always chooses the process that has the shortest expected remaining time. When a new process joins the ready queue, it may in fact have a shorter remaining time than the currently running process. Accordingly, the scheduler may preempt whenever a new process becomes ready. As with SPN, the scheduler must have an estimate of processing time to perform the selection function, and there is a risk of starvation of longer processes.

LCN does not have the bias in favour of long processes found in FCFS. Unlike round robin, no additional interrupts are generated, reducing overhead. On the other hand, elapsed service times must be recorded, contributing to overhead. LCN should also give superior turnaround time performance to STG, because a short job is given immediate perference to a running longer job.

(iii) Round Robin Policy – A straightforward way to reduce the penalty that short jobs suffer with FCFS is to use preemption based on a clock. The simplest such policy is round robin. A clock interrupt is generated

at periodic intervals. When the interrupt occurs, the currently running proce is placed in the ready queue, and the next ready job is selected on a FCF basis. This technique is also known as ***time slicing***, because each process given a time slice before being preempted.

This is the simplest method which contains the ready processes in o single queue and dispatches them one by one. This policy treats all the process in same manner and thus, it is extremely fair, but if the number of users is ve high, the response time deteriorate for online processes that need quick attenti

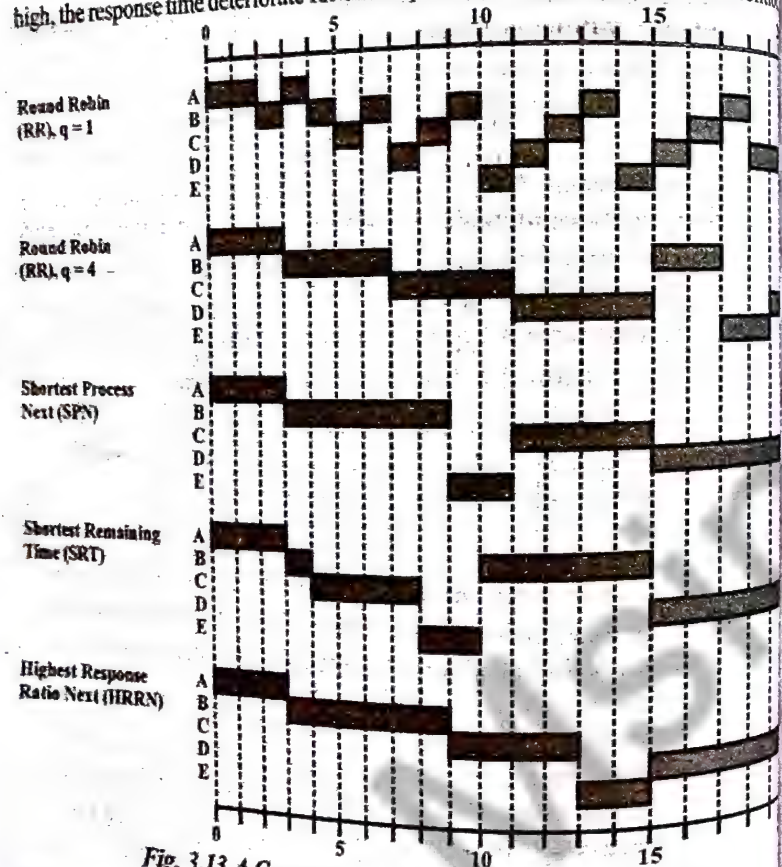


Fig. 3.13 A Comparison of Scheduling Policies

Fig. 3.13 and table 3.1 show the results for our example using time quant q of 1 and 4 time units. Note that process E, which is the shortest job, enjo significant improvement for a time quantum of 1.

(iv) Highest-response Ratio Next – Brinch Hansen developed t highest-response ratio next (HRN) strategy that corrects some of the weakne is STG policy particularly the excessive bias against longer jobs and the excess favoritism toward short new jobs. HRN is a non-preemptive schedul discipline in which the priority of each job is a function not only of the jo

service time but also of the amount of time the job has been waiting for service.Once a job gets the CPU, it runs to completion. Dynamic priorities in HRN are calculated according to the formula –

$$\text{Priority} = \frac{\text{Time waiting} + \text{Service time}}{\text{Service time}}$$

Because the service time appears in the denominator, shorter jobs will get preference. But because time waiting appears in the numerator, longer jobs that have been waiting will also be given favorable treatment. Note that the sum,

Time waiting + Service time is the system's response time to the job if the job were to be initated immediately.

Q.26. Write short note on real-time scheduling. ***(R.G.P.V., Dec. 2006)***

Ans. Real-time scheduling is one of the most active areas of research in computer science. In a survey of real-time scheduling algorithms, observes that the various scheduling approaches depend on – (i) whether a system performs schedulability analysis, (ii) if it does, whether it is done statically or dynamically, and (iii) whether the result of the analysis itself produces a schedule or plan according to which tasks are dispatched at run time. Based on these considerations, the following classes of algorithms are –

(i) Static Table-driven Approaches – These perform a static analysis of feasible schedules of dispatching. The result of the analysis is a schedule that determines, at run time, when a task must begin execution.

(ii) Static Priority-driven Preemptive Approaches – Again, a static analysis is performed, but no schedule is drawn up. Rather, the analysis is used to assign priorities to tasks, so that a traditional priority-driven preemptive scheduler can be used.

(iii) Dynamic Planning-based Approaches – Feasibility is determined at run time rather than offline prior to the start of execution (statically). An arriving task is accepted for execution only if it is feasible to meet its time constraints. One of the results of the feasibility analysis is a schedule or plan that is used to decide when to dispatch this task.

(iv) Dynamic Best Effort Approaches – No feasibility analysis is performed. The system tries to meet all deadlines and aborts any started process whose deadline is missed.

NUMERICAL PROBLEMS

Prob.1. Consider the following set of processes –

Process	*Processing Time*
A	*3*
B	*5*
C	*2*
D	*5*
E	*5*

Develop a Gantt chart and calculate the average waiting time using –
(i) FCFS (ii) SJF (iii) Round robin (q = 1).

(R.G.P.V., Dec. 2010)

Sol. (i) FCFS – The Gantt chart is as follows –

A	B	C	D	E
0–3	3–8	8–10	10–15	15–20

Waiting time for A = 0
Waiting time for B = 3
Waiting time for C = 8
Waiting time for D = 10
Waiting time for E = 15

Hence, average waiting time $= \frac{0+3+8+10+15}{5} = 7.2$ ms

(ii) SJF – The Gantt chart is as follows –

C	A	B	D	E
0–2	2–5	5–10	10–15	15–20

Waiting time for A = 2
Waiting time for B = 5
Waiting time for C = 0
Waiting time for D = 10
Waiting time for E = 15

Hence, average waiting time $= \frac{2+5+0+10+15}{5} = 6.4$ ms

(iii) Round Robin (q = 1) – The Gantt chart is as follows –

A	B	C	D	E	A	B	C	D	E	A	B	D	E	B	D	E	B	D	E
0–1	1–2	2–3	3–4	4–5	5–6	6–7	7–8	8–9	9–10	10–11	11–12	12–13	13–14	14–15	15–16	16–17	17–18	18–19	19–20

Waiting time for A = 8
Waiting time for B = 13
Waiting time for C = 6
Waiting time for D = 14
Waiting time for E = 15

Hence, average waiting time $= \frac{8+13+6+14+15}{5} = 11.2$ ms

Prob.2. Suppose that the given ahead processes arrive for execution at time indicated –

Process	*Arrival Time*	*Burst Time*
P_1	*0.0*	*8*
P_2	*0.4*	*4*
P_3	*1.0*	*1*

Calculate average turn around time, average waiting time and throughput –
(i) FCFS (ii) SRTF (iii) Non-preemptive SJF.

(R.G.P.V., June 2011)

Sol. (i) FCFS – The Gantt chart for the given set of processes is as follows –

P_1	P_2	P_3
0–8	8–12	12–13

Turnaround time for process $P_1 = 8 - 0 = 8$
Turnaround time for process $P_2 = 12 - 0.4 = 11.6$
Turnaround time for process $P_3 = 13 - 1 = 12$

Therefore, average turnaround time $= \frac{8+11.6+12}{3} = \frac{31.6}{3} = \mathbf{10.53\ ms}$

Waiting time for process $P_1 = 0 - 0.0 = 0$
Waiting time for process $P_2 = 8 - 0.4 = 7.6$
Waiting time for process $P_3 = 12 - 1.0 = 11$

Therefore, average waiting time $= \frac{0+7.6+11}{3} = \frac{18.6}{3} = \mathbf{6.2\ ms}$

Three processes executed in 13 time units. So throughput is **3/13.**

(ii) SRTF – The Gantt chart for the given set of processes is as follows –

P_1	P_2	P_3	P_2	P_1

0 0.4 1 2 5.4 13

Turnaround time for process $P_1 = 13 - 0 = 13$

Turnaround time for process $P_2 = 5.4 - 0.4 = 5$

Turnaround time for process $P_3 = 2 - 1 = 1$

Therefore, average turnaround time $= \dfrac{13+5+1}{3} = \dfrac{19}{3} = \mathbf{6.33\ ms}$

Waiting time for process $P_1 = 0 - 0.0 + 5.4 - 0.4 = 5$

Waiting time for process $P_2 = 0.4 - 0.4 + 2 - 1 = 1$

Waiting time for process $P_3 = 1 - 1.0 = 0$

Therefore, average waiting time $= \dfrac{5+1+0}{3} = \dfrac{6}{3} = \mathbf{2\ ms}$

Three processes executed in 13 time units. So throughput is 3/13.

(iii) Non-preemptive SJF – The Gantt chart for the given set of processes is as follows –

P_1	P_3	P_2

0 8 9 13

Turnaround time for process $P_1 = 8 - 0 = 8$

Turnaround time for process $P_2 = 13 - 0.4 = 12.6$

Turnaround time for process $P_3 = 9 - 1 = 8$

Therefore, average turnaround time $= \dfrac{8+12.6+8}{3} = \mathbf{9.53\ ms}$

Waiting time for process $P_1 = 0 - 0.0 = 0$

Waiting time for process $P_2 = 9 - 0.4 = 8.6$

Waiting time for process $P_3 = 8 - 1.0 = 7$

Therefore, average waiting time $= \dfrac{0+8.6+7}{3} = \dfrac{15.6}{3} = \mathbf{5.2\ ms}$

Three processes executed in 13 time units. So throughput is **3/13**.

Prob.3. Assume you have the following jobs to execute with one processor –

Process	*Arrival Time*	*Burst Time*
P_1	*0.0*	*8*
P_2	*0.4*	*4*
P_3	*1.0*	*1*

Calculate average turn around time using SJF preemptive scheduling algorithm. ***(R.G.P.V., Dec. 2014)***

Sol. Refer Prob.2 (ii).

Prob.4. Assume you have the following jobs to execute with one processor –

Job	*Execution Time*	*Arrival Time*	*Priority*
0	*80*	*0*	*2*
1	*25*	*10*	*4*
2	*15*	*20*	*3*
3	*20*	*40*	*4*
4	*45*	*50*	*1*

Using shortest job first, priority and shortest-remaining-time-first scheduling. Draw Gantt chart and calculate average waiting and turnaround time. ***(R.G.P.V., Dec. 2008)***

***Sol.* Shortest Job First (SJF)** – Here, according to the execution time and arrival time, we can draw the Gantt chart as follows –

J_0	J_2	J_3	J_1	J_4

0 80 95 115 140 185

Average Waiting Time in SJF – Seeing the Gantt chart for SJF then,

Waiting time for job $J_0 = 0$

Waiting time for job $J_1 = 115 - 10 = 105$

Waiting time for job $J_2 = 80 - 20 = 60$

Waiting time for job $J_3 = 95 - 40 = 55$

Waiting time for job $J_4 = 140 - 50 = 90$

Average waiting time $= \dfrac{0+105+60+55+90}{5} = \mathbf{62\ ms}$

Turnaround Time for SJF –

Turnaround time for job $J_0 = 80$

Turnaround time for job $J_1 = 140 - 10 = 130$

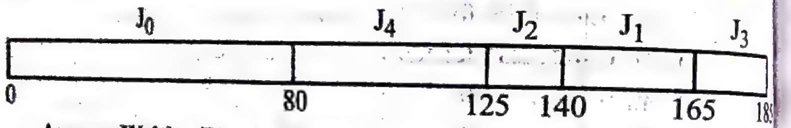
Turnaround time for job $J_2 = 95 - 20 = 75$

Turnaround time for job $J_3 = 115 - 40 = 75$

Turnaround time for job $J_4 = 185 - 50 = 135$

$$\text{Average turnaround time} = \frac{80+130+75+75+135}{5} = \frac{495}{5} = 99 \text{ ms}$$

Priority Scheduling – Here, it is assumed that the scheduling is nonpremptive, and a smaller number has higher priority. Hence, the Gantt chart is as follows –



Average Waiting Time in Priority Scheduling – Seeing the Gantt chart for priority scheduling then,

Waiting time for job $J_0 = 0$

Waiting time for job $J_1 = 140 - 10 = 130$

Waiting time for job $J_2 = 125 - 20 = 105$

Waiting time for job $J_3 = 165 - 40 = 125$

Waiting time for job $J_4 = 80 - 50 = 30$

$$\text{Average waiting time} = \frac{0+130+105+125+30}{5} = \frac{390}{5} = 78 \text{ ms}$$

Average Turnaround Time for Priority Scheduling –

Turnaround time for job $J_0 = 80$

Turnaround time for job $J_1 = 165 - 10 = 155$

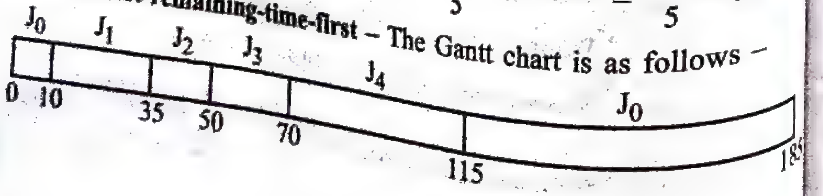
Turnaround time for job $J_2 = 140 - 20 = 120$

Turnaround time for job $J_3 = 185 - 40 = 145$

Turnaround time for job $J_4 = 125 - 50 = 75$

$$\text{Average turnaround time} = \frac{80+155+120+145+75}{5} = \frac{575}{5} = 115 \text{ ms}$$

Shortest-remaining-time-first – The Gantt chart is as follows –



Average Waiting Time in Shortest-remaining-time-first –

Waiting time for job $J_0 = 115 - 10 = 105$

Waiting time for job $J_1 = 10 - 10 = 0$

Waiting time for job $J_2 = 35 - 20 = 15$

Waiting time for job $J_3 = 50 - 40 = 10$

Waiting time for job $J_4 = 70 - 50 = 20$

$$\text{Average waiting time} = \frac{105+0+15+10+20}{5} = \frac{150}{5} = 30 \text{ ms}$$

Average Turnaround Time for Shortest-job-remaining-first –

Turnaround time for job $J_0 = 185 - 0 = 185$

Turnaround time for job $J_1 = 35 - 10 = 25$

Turnaround time for job $J_2 = 50 - 20 = 30$

Turnaround time for job $J_3 = 70 - 40 = 30$

Turnaround time for job $J_4 = 115 - 50 = 65$

$$\text{Average Turnaround Time} = \frac{185+25+30+30+65}{5} = \frac{335}{5} = 67 \text{ ms}$$

Prob.5. Assume that the following jobs are to be executed on one processor.

Job	*Execution Time*	*Arrival Time*	*Priority*
0	*80*	*0*	*2*
1	*25*	*10*	*4*
2	*15*	*20*	*3*
3	*20*	*30*	*4*
4	*45*	*40*	*1*

Using shortest job first, priority and shortest-remaining-time-first scheduling. Draw Gantt chart and calculate average waiting and turn around time. ***(R.G.P.V., Dec. 2013)***

Sol. Similar as Prob.4.

Prob.6. With reference to the following set of processes, determine average waiting time and average turnaround time. Using the following scheduling algorithms –

(i) Shortest remaining time first (ii) Priority based (Preemptive).

Process	*Arrival Time*	*CPU Burst*	*Priority*
P_1	*0*	*24*	*5*
P_2	*3*	*7*	*3*
P_3	*5*	*6*	*2*
P_4	*10*	*10*	*1*

(R.G.P.V., May 2018)

Sol. (i) Shortest Remaining Time First (SRTF) – The Gantt chart for the given set of processes is as follows –

P_1	P_2	P_3	P_4	P_1

0 3 10 16 26 47

Waiting time for process $P_1 = 26 - 3 = 23$
Waiting time for process $P_2 = 3 - 3 = 0$
Waiting time for process $P_3 = 10 - 5 = 5$
Waiting time for process $P_4 = 16 - 10 = 6$

$$\text{Therefore, average waiting time} = \frac{23+0+5+6}{4} = \frac{34}{4} = \mathbf{8.5\ ms}$$

Turnaround time for process $P_1 = 47 - 0 = 47$
Turnaround time for process $P_2 = 10 - 3 = 7$
Turnaround time for process $P_3 = 16 - 10 = 6$
Turnaround time for process $P_4 = 26 - 16 = 10$

$$\text{Therefore, average turnaround time} = \frac{47+7+6+10}{4} = \frac{70}{4} = \mathbf{17.5\ ms}$$

(ii) Priority Based (Preemptive) – The Gantt chart for the given set of processes is as follows –

P_4	P_3	P_2	P_1

0 10 16 23 47

Waiting time for process $P_1 = 23 - 0 = 23$
Waiting time for process $P_2 = 16 - 3 = 13$
Waiting time for process $P_3 = 10 - 5 = 5$
Waiting time for process $P_4 = 0$

$$\text{Average waiting time} = \frac{23+13+5+0}{4} = \frac{41}{4} = \mathbf{10.25\ ms}$$

Turnaround time for process $P_1 = 47 - 0 = 47$
Turnaround time for process $P_2 = 23 - 3 = 20$
Turnaround time for process $P_3 = 16 - 5 = 11$
Turnaround time for process $P_4 = 10 - 10 = 0$

$$\text{Average turnaround time} = \frac{47+20+11}{4} = \frac{78}{4} = \mathbf{19.5\ ms}$$

Prob.7. Consider the following jobs –

Job	*Burst Time*
1	*10*
2	*29*
3	*3*
4	*7*
5	*12*

Consider the FCFS, SJF and RR (quantum = 10) scheduling algorithms for this set of processes. Which algorithm would give the minimum average waiting time ? ***(R.G.P.V., June 2009)***

***Sol.* FCFS (First-come-first-serve)** – Consider that all the five processes arrive at time 0, with the length of the CPU-burst time given in milliseconds –

Process	*Burst time*
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

If the processes arrive in the order P_1, P_2, P_3, P_4, P_5, the result is as shown in Gantt chart

P_1	P_2	P_3	P_4	P_5

0 10 39 42 49 61

Waiting time for process P_1 = 0
Waiting time for process P_2 = 10
Waiting time for process P_3 = 39
Waiting time for process P_4 = 42
Waiting time for process P_5 = 49
Total waiting time = 140
So, Average waiting time = 140/5 = **28 ms**

SJF (Shortest Job First) – In this algorithm the shortest burst time process is executed first.

The Gantt chart for SJF scheduling is as follows –

P_3	P_4	P_1	P_5	P_2

0 3 10 20 32 61

Waiting time for process P_1 = 10
Waiting time for process P_2 = 32
Waiting time for process P_3 = 0
Waiting time for process P_4 = 3
Waiting time for process P_5 = 20
Total waiting time = 65
Average waiting time = 65/5 = **13 ms**

RR (Round Robin) – The quantum = 10 ms is given. The Gantt chart for RR scheduling is as follows –

P_1	P_2	P_3	P_4	P_5	P_2	P_5	P_2

0 10 20 23 30 40 50 52 61

Waiting time for process P_1 = 0
Waiting time for process P_2 = 52 – 50 + 40 – 20 + 10 = 32
Waiting time for process P_3 = 20
Waiting time for process P_4 = 23
Waiting time for process P_5 = 50 – 40 + 30 = 40
Total waiting time = 115 ms
Average waiting time = 115/5 = **23 ms**

Thus, SJF algorithm gives the minimum average waiting time, viz. 13 ms.

Prob.8. Consider the following set of processes, with the length of the CPU-burst time given in milliseconds –

Process	*Burst Time*	*Priority*
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are asumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

(i) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non-preemptive priority and RR (quantum = 1) scheduling.

(ii) What is the turnaround time of each process for each of the scheduling algorithms ?

(iii) What is the waiting time of each process for each of the scheduling algorithms ?

(iv) Which of the scheduling in part a results in the minimal average waiting time ?

(R.G.P.V., Dec. 2012)

Sol. (i) Gantt Charts –

FCFS – The Gantt chart is as follows –

P_1	P_2	P_3	P_4	P_5

0 10 11 13 14 19

SJF – The Gantt chart is as follows –

P_2	P_4	P_3	P_5	P_1

0 1 2 4 9 19

Non-preemptive Priority – The Gantt chart is a follows –

P_2	P_5	P_1	P_3	P_4

0 1 6 16 18 19

RR (Quantum = 1) – The Gantt chart is as follows –

P_1	P_2	P_3	P_4	P_5	P_1	P_3	P_5	P_1	P_5	P_1	P_5	P_1	P_5	P_1	P_1	P_1	P_1	P_1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

(ii) Turnaround Time –

FCFS –

Turnaround time for process P_1 = 10
Turnaround time for process P_2 = 11
Turnaround time for process P_3 = 13
Turnaround time for process P_4 = 14
Turnaround time for process P_5 = 19

SJF –

Turnaround time for process P_1 = 19
Turnaround time for process P_2 = 1
Turnaround time for process P_3 = 4
Turnaround time for process P_4 = 2
Turnaround time for process P_5 = 9

Non-preemptive Priority –

Turnaround time for process P_1 = 16
Turnaround time for process P_2 = 1
Turnaround time for process P_3 = 18
Turnaround time for process P_4 = 19
Turnaround time for process P_5 = 6

Round Robin –

Turnaround time for process P_1 = 19
Turnaround time for process P_2 = 2
Turnaround time for process P_3 = 7
Turnaround time for process P_4 = 4
Turnaround time for process P_5 = 14

(iii) Waiting Time –

FCFS –

Waiting time for process $P_1 = 0$
Waiting time for process $P_2 = 10$
Waiting time for process $P_3 = 11$
Waiting time for process $P_4 = 13$
Waiting time for process $P_5 = 14$

SJF –

Waiting time for process $P_1 = 9$
Waiting time for process $P_2 = 0$
Waiting time for process $P_3 = 2$
Waiting time for process $P_4 = 1$
Waiting time for process $P_5 = 4$

Non-preemptive Priority –

Waiting time for process $P_1 = 6$
Waiting time for process $P_2 = 0$
Waiting time for process $P_3 = 16$
Waiting time for process $P_4 = 18$
Waiting time for process $P_5 = 1$

Round Robin –

Waiting time for process $P_1 = 9$
Waiting time for process $P_2 = 1$
Waiting time for process $P_3 = 5$
Waiting time for process $P_4 = 3$
Waiting time for process $P_5 = 9$

(iv) Average Waiting Time –

FCFS –

$$\text{Average waiting time} = \frac{0+10+11+13+14}{5} = 9.6$$

SJF –

$$\text{Average waiting time} = \frac{9+0+2+1+4}{5} = 3.2$$

Non-preemptive Priority –

$$\text{Average waiting time} = \frac{6+0+16+18+1}{5} = 8.2$$

Round Robin –

$$\text{Average waiting time} = \frac{9+1+5+3+9}{5} = 5.4$$

Thus, the average waiting of SJF scheduling is minimum in compare to FCFS, RR and non-preemptive scheduling.

Prob.9. Consider the following set of processes, with the length of the CPU burst given in milliseconds –

Process	*Burst Time*
P_1	10
P_2	1
P_3	2
P_4	1
P_5	5

Calculate avg. turnaround time and avg. waiting time for round robin algorithm. (Assume quantum = 1). ***(R.G.P.V., June 2010)***

Sol. Refer Prob.8.

ALGORITHMS EVALUATION, SYSTEM CALLS FOR PROCESS MANAGEMENT, MULTIPLE PROCESSOR SCHEDULING, CONCEPT OF THREADS

Q.27. Describe queuing algorithm evaluation model.

Ans. The processes that are run on many systems vary from day-to-day, so there is no static set of processes.

The computer system is described as a network of servers. Each server has a queue of waiting processes. The CPU is a server with its ready queue, as is the I/O system with its device queues. Knowing arrival rates and service rates, we can compute utilization, average queue length, average wait time, and so on. This area of study is called queuing-network analysis.

Let n be the average queue length, let W be the average waiting time in the queue, and let λ be the average arrival rate for new processes in the queue. Then, we expect that during the time W that a process waits, λ×W new process will arrive in the queue. If the system is in a steady state, then the number of processes leaving the queue must be equal to the number of processes that arrive. Thus,

$$n = \lambda \times W$$

This equation is known as ***Little's formula***. This equation is particularly useful because it is valid for any scheduling algorithm and arrival distribution. This formula is used to compute one of the three variable, if we know the other two. For example, if we know that seven processes arrive every second, and that there are normally 14 processes in the queue, then we can compute the average waiting time per process as 2 seconds.

Queuing analysis can be useful in comparing scheduling algorithms but it also has limitations. The classes of algorithms and distributions that can be handled are fairly limited. The mathematics of complicated algorithms or distributions can be difficult to work with. Thus, arrival and service distributions are often defined in unrealistic, but mathematically tractable ways.

Q.28. Explain about deterministic algorithm evaluation model.

Ans. Analytic evaluation is one of the class of evaluation methods. It uses the given algorithm and the system workload to produce a formula or number that evaluates the performance of the algorithm for that workload. Deterministic modeling is a type of analytic evaluation. This method takes a particular predetermined workload and defines the performance of each algorithm for that workload.

Example – Consider the following processes with their CPU burst time in millisecond.The arriving order is P_1, P_2, P_3, P_4 and all arrive at time 0.

Process	*CPU Burst Time*
P_1	8
P_2	10
P_3	3
P_4	5

Then Gantt chart for FCFS is –

P_1	P_2	P_3	P_4

0 8 18 21 26

$$\text{The average waiting time} = \frac{0+8+18+21}{4} = \frac{47}{4} = 11.75 \text{ ms}$$

Similarly, the Gantt chart for SJF

P_3	P_4	P_1	P_2

0 3 8 16 26

$$\text{The average waiting time} = \frac{8+16+0+3}{4} = \frac{27}{4} = 6.75 \text{ ms}$$

Similarly, the Gantt chart for RR, if quantum = 6 ms

P_1	P_2	P_3	P_4	P_1	P_2

0 6 12 15 20 22 26

$$\text{The average waiting time} = \frac{14+16+12+15}{4} = \frac{57}{4} = 14.25 \text{ ms}$$

So, in this case , the SJF gives minimum average waiting time, and RR gives maximum average waiting time. The FCFS gives intermediate average waiting time.

Q.29. Describe simulation algorithm evaluation method.

Ans. Simulation gives more accurate evaluation of scheduling algorithms. It involves programming a model of the computer system. Software data structures represent the major components of the system.The simulator has a variable representing a clock, as this variable is increased, the simulator modifies the system state to reflect the activities of the devices, the processes and the schedulers. As the simulation executes, statistics that indicate algorithm performance are gathered and printed.

The data to drive the simulation can be generated in several ways. The most common method uses a random-number generator, which is programmed to generate processes, CPU-burst times, arrivals, departures and so on, according to probability distributions. The distributions may be defined mathematically or empirically.

Simulations can be expensive, however, after requiring hours of computer time. A more detailed simulation provides more accurate results, but also requires more computer time.

Also a trace tape can require large amounts of storage space. Finally, the design, coding, and debugging of the simulator can be a major task.

Q.30. Write short note on system calls for process management.

Ans. As we know that the system calls provide the interface between the operating system and a process. For the process management the system call supports the concept such as end and abort of the process, load and execute, create process and terminate process, get process attributes and set process attributes, wait for time, wait event and signal event, allocate and free memory.

A running program needs to be able to halts its execution sometimes, either normally (end) or abnormally (abort). When a system call is made to terminate the current running program abnormally, or if the program is running into a problem and causes an error trap, which results in the consumption of dump of memory and an error message is generated. This dump is written to disk and examined by a debugger to determine the cause of the problem. The operating system helps to transfer the control to the command interpreter, either in normal or abnormal circumstances. In a batch system, the command interpreter usually terminates the entire job and continues with the next job. In case of the error, the program discovers an error and wants to terminate abnormally, and it also define an error level.

A program which is executing the process may want to load and execute another program. It allows the command interpreter to execute a program as directed, a user command, click by mouse or a batch command.

When any new program terminates, the control returns to the existing program. The image of the existing program should be saved in the memory, thus, it is the effective mechanism for one program to call another program. If both the programs running concurrently, the new job or process should be created to be multiprogrammed.

Whenever a new job or process or, even a set of jobs or processes are created, we must be able to control their executions. This control requires the ability to determine and reset the attributes of a job or process, while including the priority of the job's, their maximum allowable execution time, this process is called as get process attributes and set process attributes. We may also terminate a job or process that we created, if we find that it is incorrect or no longer needed.

On the creation of new jobs or processes, we have to wait for them to finish their execution. It must be required to wait for a certain amount of time that is called wait time. Moreover, sometimes we wait for a specific event to occur, that is called wait event. The jobs or processes should then signal when that event has occurred, that is called signal event.

Q.31. What do you mean by multiprocessor scehduling ?

Ans. When system consists of multiple processors then multiprocessor scheduling is required to determine which process will run next and on which processor.

There may be two types of processes, those run on multiple processor system. One, those are independent to each other, just like in time sharing system each process is of different user. Other are related to each other. So for different purposes different type of multiprocessor scheduling is used.

Q.32. What are threads ? What resources are used when a thread is created ? How do they differ from those used when a process is created ?

(R.G.P.V., Dec. 2005, 2016)

Or

What resources are used when a thread is created ? How do they differ from those used when a process is created ? ***(R.G.P.V., Dec. 2015)***

Or

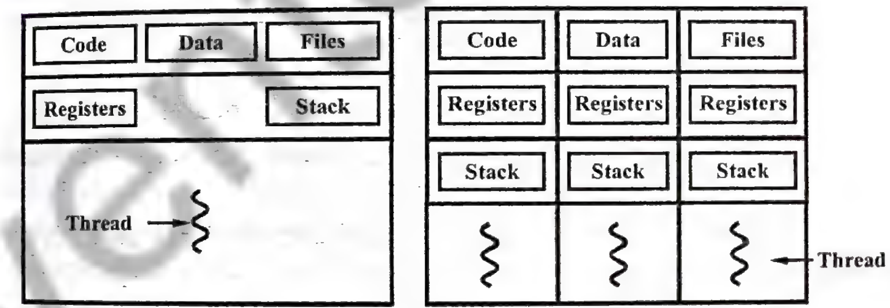
What are threads ? What resources are used when a thread is created ?

(R.G.P.V., May 2018)

Ans. In traditional operating systems, each process has an address space and a single thread of control. In fact, that is almost the definition of a process.

Nevertheless there are frequently situations in which it is desirable to have multiple threads of control in the same address space running in quasi-parallel as though they were separate processes.

A thread sometimes called a ***light weight process (LWP)***, is basic unit of CPU utilization, it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. A ***heavy weight (or traditional) process*** has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time. The difference between a traditional single-threaded process and a multithreaded process is shown in fig. 3.14.



Single-threaded **Multithreaded**

Fig. 3.14 Single and Multithread Processes

Many software packages that run on modern desktop PCs are multithreaded. An application typically is implemented as a separate process with several threads of control. For example, a Web browser might have one thread for display image or text while another thread for retrieve data from the network. A word processor may have a thread for displaying graphics, another thread for reading keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.

Different threads in a process are not quite as independent as different processes. All threads have exactly the same address space, which means that they also share the same global variables. Since every thread can access every memory address whithin the process address space, one thread can read, write, or even completely wipe out another thread's stack. There is no protection between threads because – (i) it is impossible, and (ii) it should not be necessary. Different processes, which may be from different users and which may hostile to one another, a process is always owned by a single user, who has presumably created multiple threads so that they can cooperate. In addition to sharing an address space, all the threads share the same set of open files, child processes, alarms, signals, etc. as shown in fig. 3.15.

The items in the first column are process properties, not thread properties.

Like a traditional process, a thread can be in any one of several states-running, blocked, ready or terminated. A thread which currently has CPU is called active. A blocked thread is waiting for some event to unblock it.

Per Thread Items	Per Process Items
Program Counter	Address Space
Stack	Global Variables
Register Set	Open Files
Child Threads	Child Processes
State	Timers
	Signals
	Semaphores
	Accounting information

Fig. 3.15 Per Thread and Per Process Concepts

Q.33. Write comparison between process and threads.

(R.G.P.V., June 2010)

***Ans.* Differences** – Following are the differences between process and thread –

(i) A thread is a lightweight process with a reduced state while a process is a heavyweight.

(ii) Different threads in a process are not as independent as processes.

(iii) A process is a program in execution while a thread is a flow of control within a process.

Similarities – Following are the similarities between process and thread –

(i) Threads share the CPU as processes do.

(ii) Threads can create child threads and can block waiting for system calls to complete, just like regular processes.

(iii) Like traditional processes, threads can be in one of the several states – running, blocked, ready or terminated.

Q.34. Discuss the various types of threads.

Ans. Following are the various types of threads –

(i) User-level Thread – They are supported above the kernel and are implemented by a thread library at the user-level. The library supports for thread creation, scheduling, and management with no support from the kernel. Because the kernel is unaware of user-level threads, all thread creation and scheduling are done in user space without the need for kernel intervention. Thus, user-level threads are fast to create and manage.

However, they have drawbacks. For instance, if the kernel is single-threaded, then any user-level thread performing a blocking system call will cause the entire process to block, even if other threads are available to run within the application.

Fig. 3.16 shows the structure of user-level threads.

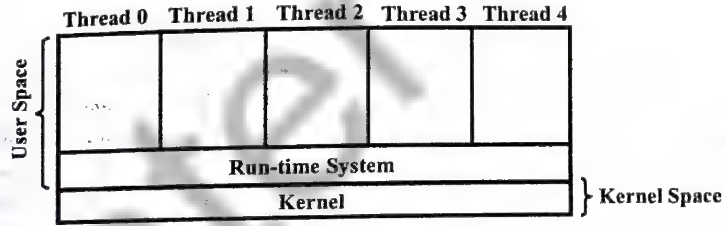


Fig. 3.16 User-level Threads

The user-level threads run on upper of run-time system. The run-time system is a collection of procedures and manages threads. Whenever any thread executes a system call, it calls a run-time system procedure.

(ii) Kernel-level Threads – They are supported directly by the operating system. The kernel performs creation, scheduling, and management of threads in kernel space. Since the thread management is done by the operating system, kernel threads are generally slower to create and manage than user threads. However, since the kernel is managing the threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. Also, in a multiprocessor environment, the kernel can schedule threads on different processors.

In the kernel-level threads, the kernel have knowledge about the threads and management of it. Here, no run-time system is needed, as shown in fig. 3.17.

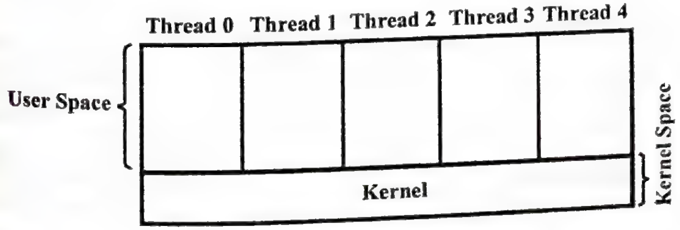


Fig. 3.17 Kernel-level Threads

Q.35. Explain multithreading models in brief.

Ans. There are many systems that provide support for both user and kernel threads, resulting in different multithreading models. There are following

types of threading implementation –

(i) ***Many-to-one Model*** – The many-to-one model maps many user-level threads to one kernel thread as shown in fig. 3.18. Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call. Also, multiple threads are unable to run in parallel on multiprocessors, because only one thread can access the kernel at a time. For example, green threads – a thread library available for Solaris 2 – uses this model.

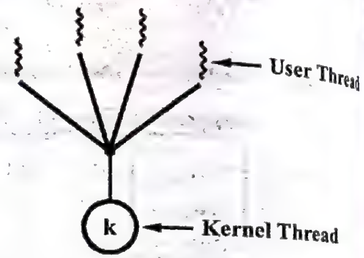


Fig. 3.18 Many-to-one Model

(ii) ***One-to-one Model*** – The one-to-one model maps each user thread to a kernel thread as shown in fig. 3.19. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call. It also allows multiple threads to run in parallel on multiprocessors. The drawback to this model is that creating a user thread requires creating the corresponding kernel thread. Because the overhead of creating kernel threads can burden the performance of an application, implementation of this model restricts the number of threads supported by the system. For example, Windows NT, Windows 2000 and OS/2 implement this model.

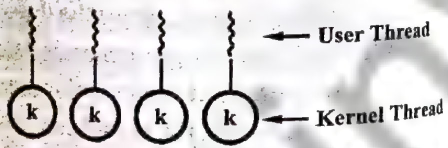


Fig. 3.19 One-to-one Model

(iii) ***Many-to-many Model*** – The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads as shown in fig. 3.20. The number of kernel threads may be specific to either a particular application or a particular machine. Whereas the many-to-one model allows the developer to create as many user threads as he wishes, true concurrency is not gained because the kernel can schedule only one thread at a time. The one-to-one model allows greater concurrency, but the developer has to be careful not to create so many threads within an application. The many-to-many model does not suffer from these drawbacks. Developers can create as many user threads as required, and the corresponding kernel threads can run in parallel on a multiprocessor. For example, Salaris 2 IRI X and Tru64 UNIX implement this model.

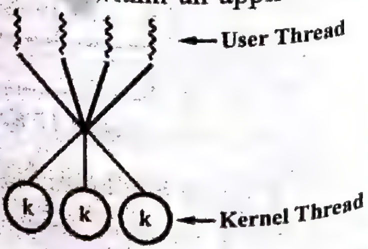


Fig. 3.20 Many-to-many Model

MEMORY MANAGEMENT – DIFFERENT MEMORY MANAGEMENT TECHNIQUES – PARTITIONING, SWAPPING, SEGMENTATION, PAGING, PAGED SEGMENTATION

Q.36. What do you mean by memory management ?

Ans. Memory is central to the operation of a computer system. It consists of a large array of words or bytes, each with its own address. In a uniprogramming system, main memory is divided into two parts – one part for the operating system and one part for the program currently being executed. In a multiprogramming system, the "user" part of memory must be further divided to accommodate multiple processes. The task of subdivision is carried out by the operating system and is known as ***memory management***.

Q.37. Explain cache memory organization.

(R.G.P.V., June 2010, Dec. 2015)

Or

Discuss cache memory organization. ***(R.G.P.V., Dec. 2010)***

Ans. If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is called as a ***cache memory***. It is placed between the CPU and main memory as shown in fig. 3.21. The cache memory access time is less than the access time of main memory by a factor of 5 to 10. The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

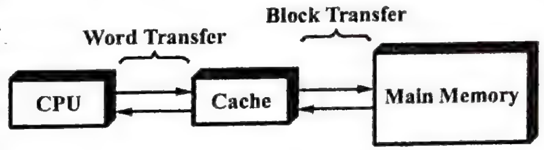


Fig. 3.21 Cache and Main Memory

The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will approach the access time of the cache. Although the cache is only a small fraction of the size of main memory, a large fraction of memory requests will be found in the fast cache memory because of the locality of reference property of programs.

The basic operation of the cache is as follows. When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to

cache memory. The block size may vary from one word to about 16 words adjacent to the one just accessed. In this way, some data are transferred to cache so that future references to memory find the required words in the fast cache memory.

Q.38. What is address binding? When we does the instruction and data binding ?

Ans. A program resides on a disk as a binary executable file. The program must be brought into memory and placed within a process for it to be executed. Depending on the memory management in use, the process may be moved between disk and memory during its execution. The collection of processes on the disk that is waiting to be brought into memory for execution forms the ***input queue.***

The general procedure is to select one of the processes in the input queue and to load that process into memory. As the process is executed, it accesses instructions and data from memory. Eventually, the process terminates and its memory space is declared available.

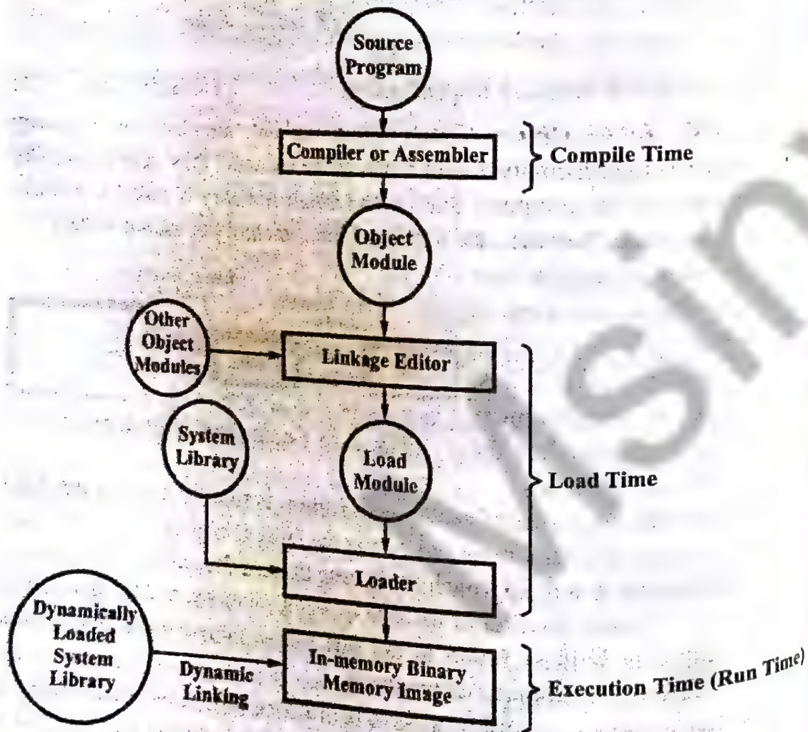


Fig. 3.22 Multistep Processing of a user Program

Many systems allow a user process to reside in any part of the physical memory. Thus, although the address space of the computer starts at 00000, the first address of the user process does not need to be 00000. This arrangement affects the addresses that the user program can use. In most cases, a user program will go through several steps – some of which may be optional before being executed (fig. 3.22). Addresses may be represented in different ways during these steps. Addresses in the source program are generally symbolic. A compiler will typically bind these symbolic addresses to relocatable addresses. The linkage editor or loader will in turn bind these relocatable addresses to absolute addresses. Each binding is a mapping from one address space to another.

The binding of instructions and data to memory addresses can be done at any step along the way –

(i) Compile Time – If we know at compile time where the process will reside in memory then absolute code can be generated. For example, if you know a priori that a user process resides starting at locations R, then the generated compiler code will start at that location and extend up from there. If after some time, the starting location changes, then it will be necessary to recompile this code. The MS-DOS.COM format programs are absolute code bound at compile time.

(ii) Load Time – If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code. Final binding is delayed until load time. If the starting address changes we need only to reload the user code to incorporate this changed value.

(iii) Run time – If the process can be moved during its execution from one memory segment to another then binding must be delayed until run time. Special hardware must be available for this scheme to work.

Q.39. Explain the difference between logical and physical address. ***(R.G.P.V., Dec. 2011)***

Or

Define logical and physical address space. ***(R.G.P.V., June 2016)***

Or

Differentiate between the physical address and logical address. ***(R.G.P.V., Dec. 2016)***

Ans. A physical address, or absolute address, is an actual location in main memory whereas a logical address is a reference to a memory location independent of the current assignment of data to memory; a translation must be made to a physical address before the memory access can be achieved.

The compile time and load time address binding methods generate same logical and physical addresses. However, the execution time address binding

results in different logical and physical addresses. In this case, the logical address is referred to as a ***virtual address***. The set of all logical addresses generated by a program is called a ***logical address space*** and the set of all physical addresses corresponding to these logical addresses is called a ***physical address space***.

The run time mapping from virtual to physical addresses is done by a hardware device called ***memory-management unit*** (MMU).

In fig. 3.23, we can see that the content of relocation register is 800. So if CPU generates 250 address, it is mapped by relocation register to 1050, by adding 800 to 250. This new address 1050 is referred as physical address.

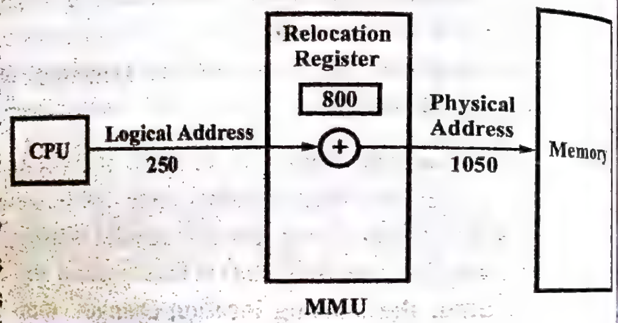


Fig. 3.23 Mapping from Logical Address to Physical Address

Q.40. What is dynamic relocation ? **(R.G.P.V., Dec. 2014)**

Ans. Dynamic relocation is used at the run time, for each instruction. It is normally done by a special piece of hardware. It is faster though somewhat more expensive. This is because, it uses a special register called base register. This register contains the value of relocation.

Q.41. Explain various memory management schemes –
(i) Single-partitioned memory management scheme
(ii) Multiple-partitioned memory management scheme
(iii) Dynamic-partitioned memory management scheme.
Or
Explain the difference between MVT and MFT. **(R.G.P.V., Dec. 2011)**
Or
Discuss memory management techniques. **(R.G.P.V., June 2017)**

Ans. (i) Single-partitioned Memory Management Scheme – This scheme does not support multiprogramming. A user can run only one program or one job or one process at a time. This scheme is basically meant for single user system, where only one process comes at a time in the memory.

The memory has two parts. One part is for operating system. The

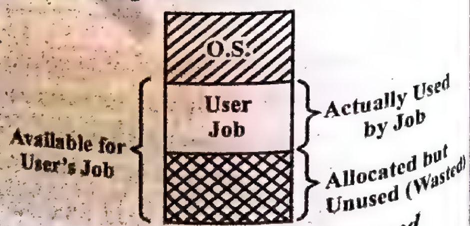
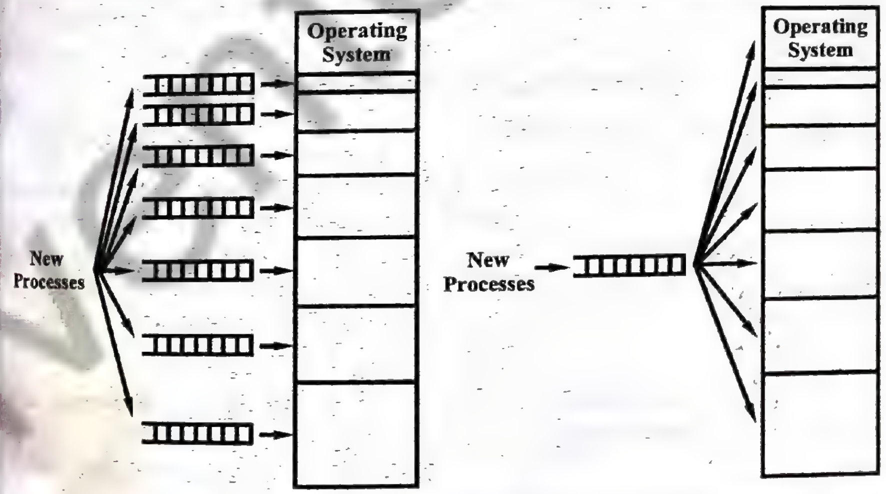


Fig. 3.24 Single-partitioned Contiguous Allocation

other part of memory is available for user program. But, only one program can reside in this partition at a time. Let us take an example as shown in fig. 3.24.

In fig. 3.24 upper part is allocated for operating system. The other part is for user job. The job actually uses only a portion of available memory, leaving an allocated but unused partition of memory.

(ii) Multiple-partitioned Memory Management Scheme – One way to support multiprogramming is to divide the available physical memory into several fixed sized partitions, each of which may be allocated to a different process. Thus, the degree of multiprogramming is bound by the number of partitions. These partitions may be of equal size or unequal size. This partitioning can be done manually by the operator when the system is started up.



(a) One Process Queue Per Partition ***(b) Single Queue***

Fig. 3.25 Memory Assignment for Fixed Partitioning

In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. This method is also known as multiprogramming with a fixed number of tasks **(MFT)**.

(iii) Dynamic-partitioned Memory Management Scheme – It is also known as multiprogramming with variable partitions or multiprogramming with variable number of tasks **(MVT)**. This approach was developed to overcome some of the difficulties with fixed partitioning.

When variable partitions are used, the number and size of the processes in memory vary dynamically. Fig. 3.26 shows how variable partitions work. Initially, only process A is in memory. Then process B and C are created or swapped in from disk. In fig. 3.26 (d) A terminates or is swapped out to disk. Then D comes and B goes out. Finally, E comes in.

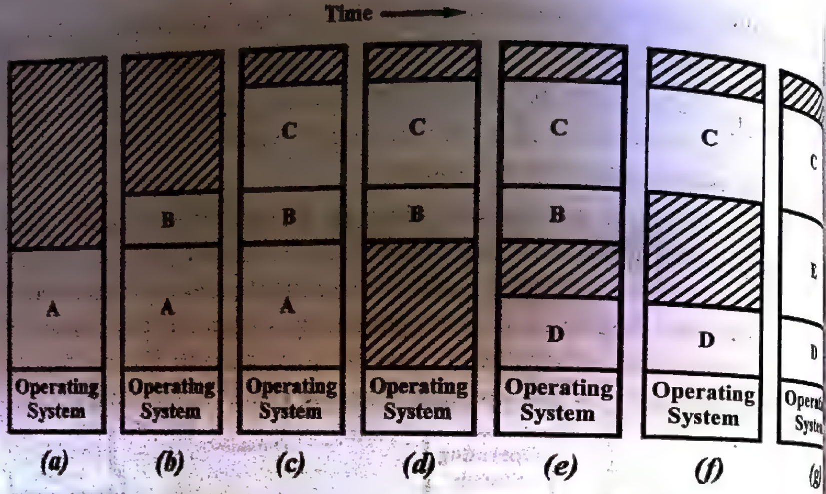


Fig. 3.26 Dynamic Memory Partitioning

Q.42. What is fragmentation ? ***(R.G.P.V., Dec. 201***

Ans. If total available memory is divided into multiple partitions of fix sized or variable sized either statically or dynamically, the unused memo space remains internal or external to the partitions. When a process com and requests for memory, its request is not satisfied while the sum of memo unused is greater than the process's request. This problem is known *fragmentation.*

Q.43. What is fragmentation ? What are its types ? How they minimized in different memory management schemes ? ***(R.G.P.V., Dec. 2004, Nov./Dec. 200***

Ans. **Fragmentation** – Refer Q.42.

Types of Fragmentation – There are two types of fragmentation –

(i) Internal Fragmentation – The phenomenon, in which there wasted space internal to a partition due to the fact that the block of data load is smaller than the partition, is referred to as internal fragmentation.

(ii) External Fragmentation – The phenomenon, in which the is wasted space external to partition due to the fact that the memory that external to all partitions becomes increasingly fragmented, is referred to external fragmentation.

When partitioning is static, i.e. fixed-size partitioning, memory is waste in each partition due to an object of smaller size than the partition itself loaded. For example, suppose there is a partition of 10,232 bytes in memo and a request comes for 10,230 bytes of memory. If this request is satisfie then 2 bytes of memory will be left unused. This is shown in fig. 3.27.

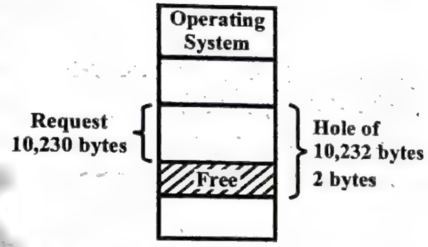


Fig. 3.27 Internal Fragmentation

Dynamic partitioning or multiprogramming with variable partitions eliminates internal fragmentation by making each partition only as large as necessary to satisfy a request for space for an object. With dynamic partitioning, the partitions are of variable length and number. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more. An example, using 64 Mbytes of main memory is shown in fig. 3.28. Initially, main memory is empty, except for the operating system in fig. 3.28 (a). The first three processes are loaded in, starting where the operating system ends and occupying just enough space for each process as in fig. 3.28 (b), (c), (d). This leaves a hole at the end of memory, that is too small for a fourth process. At some point, none of the processes in memory is ready. The operating system swaps out process 2 as in fig. 3.28 (e), which leaves sufficient room to load a new process. Now, it loads process 4 as in fig. 3.28 (f). Because process 4 is smaller than process 2, another small hole is created. Later, a point is reached at which none of the

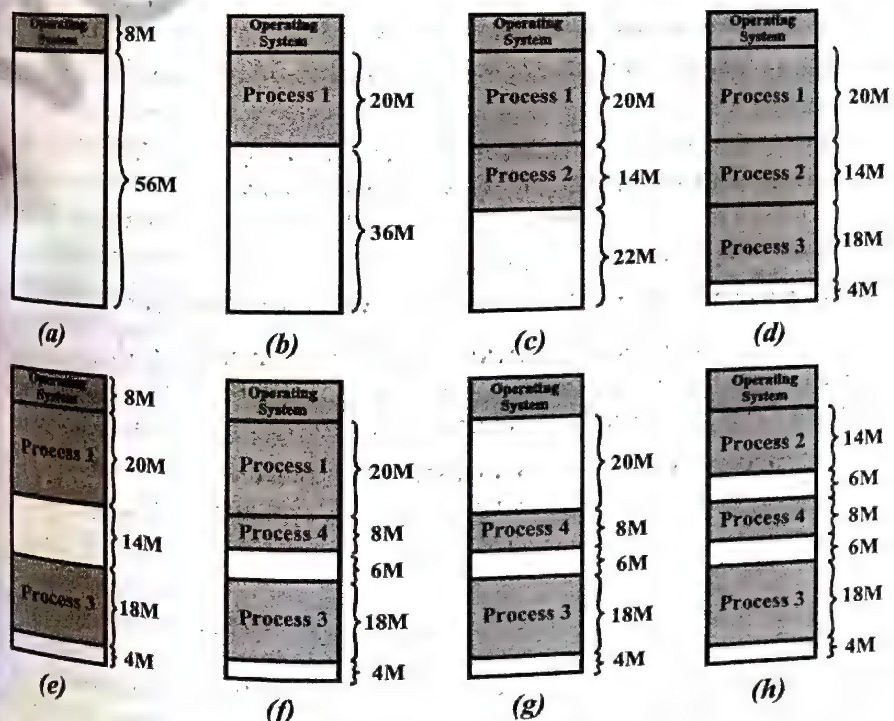


Fig. 3.28 The Effect of Dynamic Partitioning

processes in main memory is ready, but process 2, in the ready-suspend state, is available. Because, there is no sufficient room in memory for process 2, the operating system swaps process 1 out and swaps in process 2, as in fig. 3.28 (g) and (h) respectively. Thus, it leads to a situation in which there are a lot of small holes in memory, i.e. external fragmentation.

One technique for overcoming external fragmentation is compaction. From time to time, the operating system shifts the processes so that they are contiguous and so that all of the free memory is together in one block. For example, in fig. 3.28 (h), compaction will result in a block of free memory of length 16 M.

Q.44. Explan the difference between internal and external fragmentations. ***(R.G.P.V., Dec. 2011)***

Or

Describe internal and external fragmentation with illustrative examples. ***(R.G.P.V., Dec. 2012)***

Or

Differentiate between external and internal fragmentation. ***(R.G.P.V., Dec. 2015)***

Ans. Refer Q.43.

Q.45. What do you understand by contiguous memory allocation ?

Ans. Contiguous allocation of memory means that each logical object, such as program or data, is placed into a set of consecutive memory locations. In contiguous memory allocation, the memory is divided into two parts – One for the resident operating system, and other for the user processes. The operating system can be placed in either low memory or high memory. The major factor that affects this decision is the location of interrupt vector. Since the interrupt vector is often in low memory, the operating system is also placed in low memory.

There are several processes to reside in memory at the same time. The main memory is partitioned into a number of areas that are allocated among the requesting processes. This partitioning may be static or dynamic. In either case, contiguous allocation suffers from the problem of memory *fragmentation,* where portions of physical memory are unused but cannot be allocated to requestors.

Q.46. What do you mean by noncontiguous allocation of memory ?

Ans. Noncontiguous allocation means that memory is allocated in such a way that parts of single logical object may be placed in noncontiguous areas of physical memory. Address translation performed during the execution of instructions establishes the necessary correspondence between a contiguous virtual-address space and the possibly discontiguous physical addresses of locations where object item reside in physical memory at run time.

The memory management scheme called virtual memory allows execution of processes when only portions of their address space are resident in primary memory. The actively used portions of an address space are kept in memory, and others are brought in from the secondary storage when needed. Depending on whether paging or segmentation is used to manage physical memory, virtual-memory is referred to as demand paging or demand segmentation. Thus, in addition to managing the primary memory, virtual memory provides for automatic migration of portions of address spaces of active processes between primary and secondary storage.

Q.47. What are the advantages and disadvantages of contiguous and non-contiguous memory allocation ? ***(R.G.P.V., June 2017)***

Ans. (i) Contiguous Memory Allocation –

Advantages –

(a) Fast, sequential and direct access.
(b) Good performance.
(c) Minimal disk seek.

Disadvantages –

(a) File size should be defined in advance.
(b) File cannot be extended.
(c) Wastage of memory.
(d) Fragmentation.

(ii) Noncontiguous Memory Allocation –

Advantages –

(a) No external fragmentation.
(b) Allows interdepence of code and data among processes.
(c) Strongly supported virtual memory allocation.
(d) Utilization of memory.
(e) File can be extended.

Diasdvantages –

(a) Slow access.
(b) CPU overhead.
(c) Low performance then contiguous memory allocation.

Q.48. Describe first fit, best fit and worst fit strategies of memory allocation. Which one of them suffers from the problem of external fragmentation and why ? ***(R.G.P.V., Dec. 2006)***

Or

Describe first fit, best fit and worst fit strategies for disk space allocations with their merits and demerits. ***(R.G.P.V., Dec. 2013)***

Ans. The first fit, best fit, and worst fit strategies are used to select a fre hole from the set of available holes.

(i) ***First Fit*** – Allocate the first hole that is big enough. In thi scheme, searching is started at the beginning of the set of holes or where the previous first fit search ended. Searching is stopped as soon as a free hole tha is large enough is found.

(ii) ***Best Fit*** – Allocate the smallest hole that is big enough. In thi scheme, the entire list is searched until the list is kept ordered by size. Thi strategy produces the smallest leftover hole.

(iii) ***Worst Fit*** – Allocate the largest hole. In this scheme, the entire list i searched unless it is sorted by size. This strategy produces the largest leftover hol

These strategies suffer from ***external fragmentation***. As processes ar loaded and removed from memory, the free memory space is broken into littl pieces. External fragmentation exists when enough total memory space exist to satisfy a request, but it is not contiguous; storage is fragmented into a larg number of small holes. This fragmentation problem can be severe. In th worst case, we could have a block of free (i.e., wasted) memory betwee every two processes. If all this memory were in one big free block, we migh be able to run several more processes.

Q.49. Explain swapping with suitable example. (R.G.P.V., Nov./Dec. 2007)

Ans. Moving processes from main memory to disk and disk to mai memory is called ***swapping***. When a process is brought in memory, calle ***swap in***, and when a process is brought back in disk, called ***swap out***.

Swapping has traditionally been used to implement multiprogramming i systems with restrictive memory capacity or with little hardware support fo memory management. Swapping is also helpful for improving processo utilization in partitioned memory environments by increasing the ratio of read to resident process. Swapping is usually employed in memory manageme systems with contiguous allocation such as fixed and dynamically partitione memory and segmentation.

It is the work of swapper that, which process will brought back in dis when there is no space in memory and a new process want to execute. Generally priority based scheduling is used by swapper to ***swap in*** and ***swap out*** processes. The priority given to each process may depend on process size or the time required for its execution. The process that

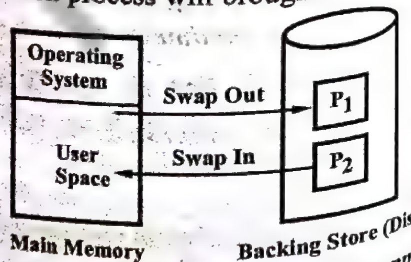


Fig. 3.29 Demonstration of Swapping

has highest priority is put first by the swapper in memory and the process that has lowest priority is put out by the swapper from memory.

The time required to swap out a process and then swap in another process is referred as ***swap time***, i.e.,

Swap time = Time to swap out a process + Time to swap in other process

The ***swap time*** highly depends on transfer rate and the size of process. If size of process is small and transfer rate is high, swap time will be very small.

Q.50. What is segmentation ?

Ans. Segmentation is a memory management scheme that divides the address space of a single process into blocks that may be placed into noncontiguous areas of memory to reduce the average size of a request.

Q.51. What advantage does segmentation offer over multiple variable partitions ? (R.G.P.V., Dec. 2015, June 2016)

Ans. Because programs are being divided into segments, the size of the object being allocated memory is smaller than it is in multiple varaible partitions. This increases the chances for filling small memory holes, decreasing central fragmentation.

Q.52. What is paging ?

Ans. Paging is a memory-management scheme that removes the requirement of contiguous allocation of physical memory. Address mapping is used to maintain the illusion of contiguity of the virtual address space of a process despite its discontiguous placement in physical memory. Physical memory is broken into fixed-size blocks called ***frames***. Logical memory is also broken into blocks of the same size called ***pages***. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.

Q.53. Explain paging and segmentation. How they are helpful in removing fragmentation ? (R.G.P.V., Dec. 2013)

Ans. Refer Q.52 and Q.50.

Q.54. Explain the principle of operation of paging scheme using suitable example.

Ans. The hardware support for paging is shown in fig. 3.30. Every address generated by the CPU is divided into two parts – a page number (p), a page offset (d). The page number is used as an index into a page table. The page

table contains the base address of each page in physical memory. The base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

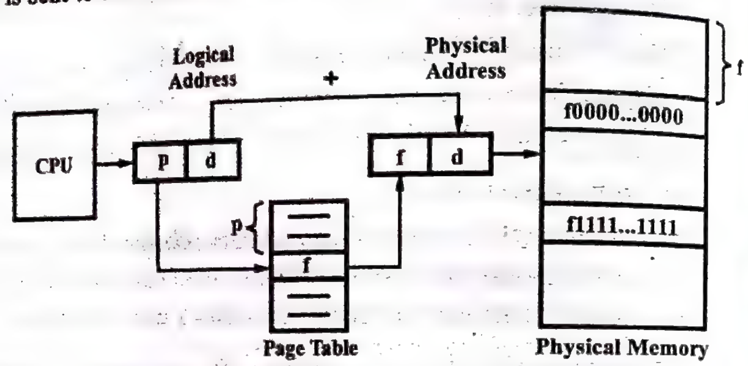


Fig. 3.30 Paging Hardware

The paging model of memory is shown in fig. 3.31. The page size is defined by the hardware. The size of page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of logical address space is 2^m, and a page size is 2^n addressing units, then the high order m – n bits of logical address indicates the page number, and the n low-order bits shows the page offset. Thus, the logical address is as follows –

Page Number	Page Offset
p	d
m – n	n

where p is an index into the page table and d is the displacement within the page.

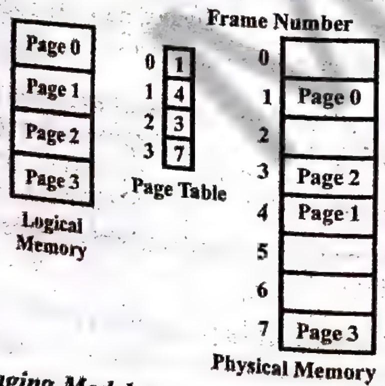


Fig. 3.31 Paging Model of Logical and Physical Memory

For example, consider the memory in fig. 3.32. Using a page size of 4 bytes and a physical memory of 32 bytes (i.e., 8 pages), we show how the user's view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 (= (5 × 4) + 0). Logical address 3 (page 0, offset 3) maps to physical address 23 (= (5 × 4) + 3). Logical address 4 is page 1, offset 0, according to page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 (= (6 × 4) + 0). Logical address 13 maps to physical address 9.

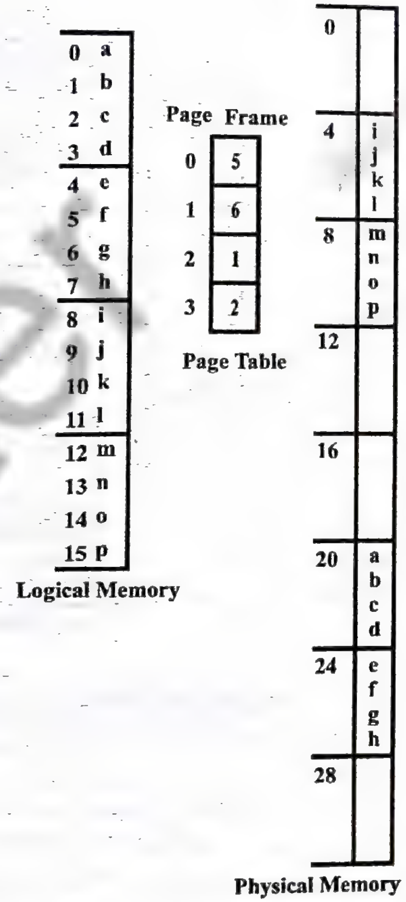


Fig. 3.32 Paging Example for a 32 byte Memory with 4 byte Pages

Q.55. Discuss various techniques for structuring the page table in paging system.

Ans. The most common techniques for structuring the page table are as follows –

(i) Hierarchical Paging – Modern computer systems support a large logical-address space from 2^{32} to 2^{64}. In such an environment, the page table itself becomes too large. For example, a system with a 32-bit logical-address space and the page size is 4 KB (2^{12}), then a page table may consist of upto 1 million entries ($2^{32}/2^{12}$). If each entry consists of 4 bytes, each process may need upto 4 MB of physical-address space for the page table. One simple solution to this problem is to divide the page table into smaller pieces.

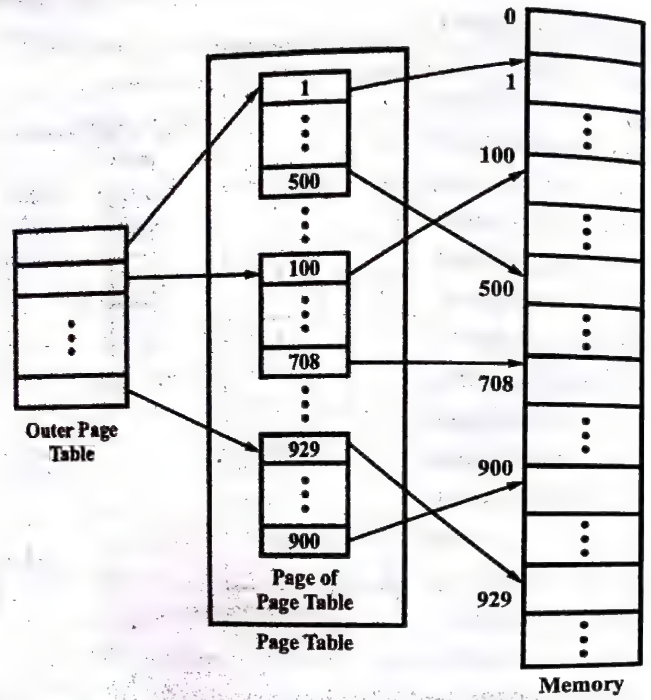


Fig. 3.33 A Two-level Page-table Scheme

The way to accomplish this division is to use a two-level paging algorithm, in which the page table itself is also paged as shown in fig. 3.33. A logical address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits. The page number is further divided into a 10-bit page number and a 10-bit page offset to page to page table. Thus, a logical address is as follows –

Page number		Page offset
p_1	p_2	d
10	10	12

where p_1 is an index into the outer page table and p_2 is the displacement within the page of the outer page table. The address translation method for this

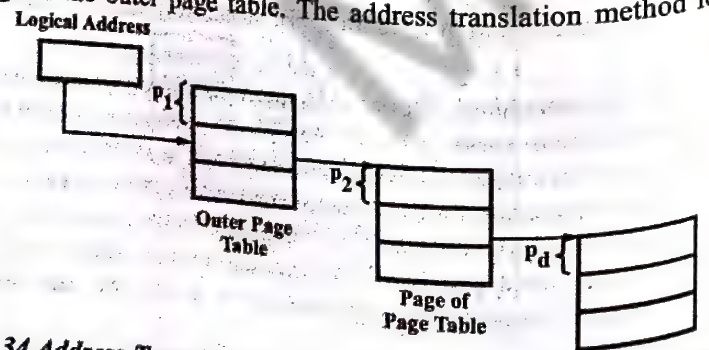


Fig. 3.34 Address Translation for a Two-level 32-bit Paging Architecture

architecture is shown in fig. 3.34. This scheme is also known as forward mapped page table. The Pentium-II uses this architecture.

(ii) Hashed Page Tables – A hashed page table is used, to handle address space larger than 32 bits, with the hash value being the virtual-page number. Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions). Each element consists of three fields – (a) the virtual page number, (b) the value of the mapped page frame, and (c) a pointer to the next element in the linked list.

The algorithm works as follows – The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared to field (a) in the first element in the linked list. If there is a match, the corresponding page frame [field (b)] is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number. This scheme is shown in fig. 3.35.

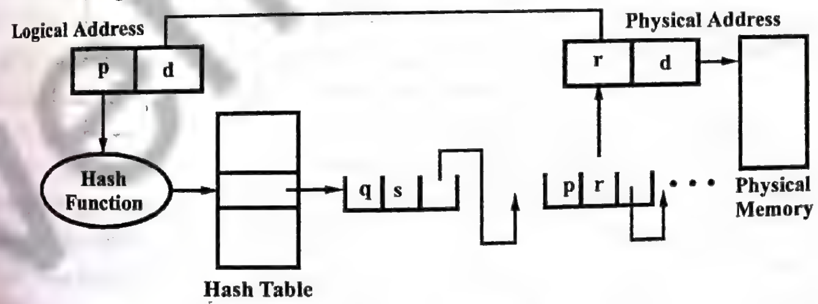


Fig. 3.35 Hashed Page Table

A variation to this scheme that is favorable for 64-bit address space has been proposed. Clustered page tables are similar to hashed page tables except that each entry in the hash table refers to several pages rather than a single page. Therefore, a single page table entry can store the mappings for multiple physical-page frames. Clustered page tables are particularly useful for sparse address spaces where memory references are noncontiguous and scattered throughout the address space.

(iii) Inverted Page Table – Usually, each process has a page table associated with it. The page table has one entry for each page that the process is using. One of the drawbacks of this method is that each page table may consist of millions of entries. These tables consume large amounts of physical memory, which is required to keep track of how the physical memory is being used.

To solve this problem, an inverted page table is used. An inverted page table has one entry for each real page (or frame) of memory. Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page. Thus, only one page table is used in the system, and having only one entry for each page of physical

memory. Fig. 3.36 shows the operation of an inverted page table. Inverted page table often requires an address-space identifier stored in each entry of the page table to ensure the mapping of a logical page for a particular process to the corresponding physical page frame. The systems using inverted page tables are 64-bit Ultra SPARC and Power PC.

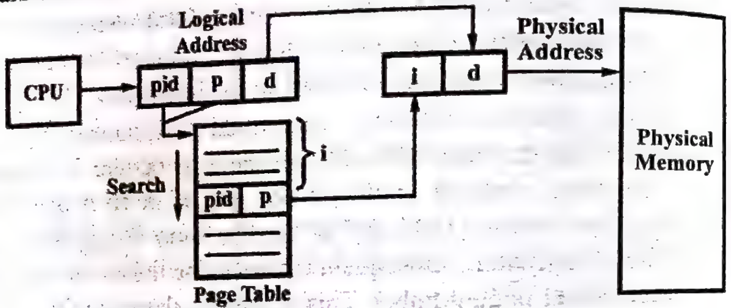


Fig. 3.36 Inverted Page Table

To illustrate this method, we describe a simplified version of the implementation of the inverted page table used in IBMRT. Each virtual address in the system consists of a triple

<process-id, page-number, offset>

Each inverted page-table entry is a pair <process-id, page number> where the process-id plays the role of the address-space identifier. When a memory reference occurs, part of the virtual address, consisting of <process-id, page-number>, is presented to the memory subsystem. The inverted page table is then searched for a match. If a match is found say at entry i, then the physical address < i, offset > is generated. If no match is found, then an illegal address has been attempted.

This scheme decreases the amount of memory needed to store each page table, but it increases the amount of time needed to search the table when a page reference occurs. To overcome this problem, a hash table is used to limit the search to one or at most a few page table entries.

Q.56. Explain the term inverted page table. ***(R.G.P.V., Dec. 201***

Ans. Refer Q.55 (iii).

Q.57. Explain the impact of page size on the overall system performance. ***(R.G.P.V., Dec. 2013)***

Ans. A minimum unit of allocation is a page. Thus, if we consider the best case where the size of the process image is taken as an exact multiple of page size, then each page frame will be fully loaded and utilized, and wastage will be 0. And if we consider the worst case where we require only 1 word to be allocated in the last page, full page frame will have to be allocated for the last

word in the program and hence, the whole page frame will be almost wholly wasted. Thus in paging, there is internal fragmentation and no external fragmentation. This is because, in paging, it cannot happen that the entire memory needed for a process is 6 pages and they are actually present, but they cannot be allocated because they scattered and not contiguous. Thus, the total average wastage = (page size –1)/2 per process, and it is higher for higher page size.

Why then not have a very small page size ? If we have very small page size, this memory wastage reduces but then as the page size reduces, you should need more number of pages per process. This is why the size of the PMP enhances. This not only results in some more memory demands, but also in higher search times for address translation in software in the hybrid and software methods, or higher charges for the hybrid and hardware methods. Hence, an optimal page size has to be selected for good performance, lower wastage and cost.

Q.58. Discuss about the hardware support for paging systems using associative memory.

Ans. Each operating system has its own methods for storing page tables. Most allocate a page table for each process. A pointer to the page table is stored with the other register values (like the instruction counter) in the process control block.

The hardware implementation of the page table is achieved in several ways. In the simplest case, the page table is implemented as a set of dedicated registers built with very high speed logic to make the paging-address translation efficient.

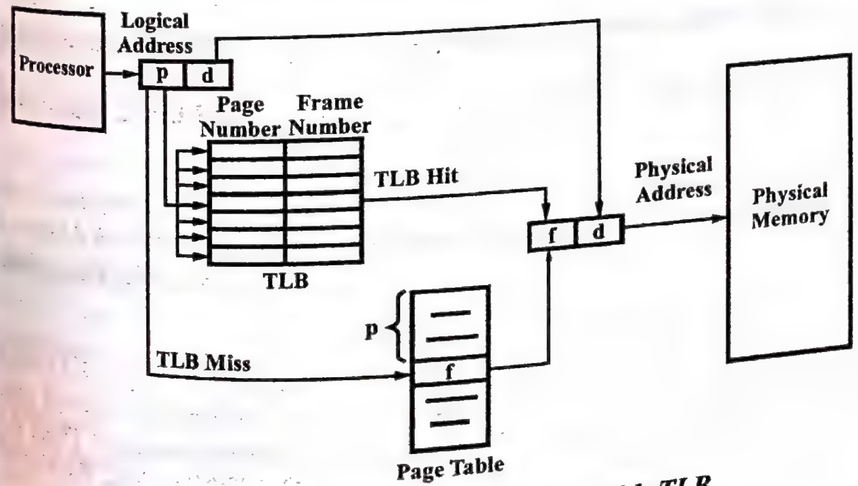


Fig. 3.37 Paging Hardware Support with TLB

The use of registers for the page table is satisfactory if the page table is reasonably small (for example, 256 entries). Most contemporary computers, however, allow the page table to be very large (for example, 1 million entries). For these computers, the use of fast registers to implement the page table is not feasible. Rather, the page table is kept in main memory, and a ***page table base register (PTBR)*** points to the page table. Changing page tables requires changing only this register. Hence, it substantially reduces context-switch time.

The problem with this approach is the time required to access a user memory location. To access location i, we must first index into the page table, using the value in the PTBR offset by the page number for i. This task requires a memory access. It provides the frame number, which is combined with the page offset to produce the actual address. We can then access the desired place in memory. Thus, two memory accesses are needed to access a byte - one for the page table entry and one for the byte. Hence, memory access is slowed by a factor of 2.

The standard solution to this problem is to use a special, small, fast-lookup hardware cache, called ***translation look-aside buffer (TLB)***. The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts – a key (or tag) and a value. The TLB is used with page tables in the following way. The TLB contains only a few of the page table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found (known as TLB hit), its frame number is immediately available and is used to access memory. If the page number is not in TLB (known as a TLB miss), a memory reference to the page table must be made.

Q.59. Define TLB with diagram. ***(R.G.P.V., Dec. 2012)***

Or

Write short note on TLB. ***(R.G.P.V., May 2018)***

Ans. Refer Q.58.

Q.60. What do you mean by inverted paging ? A RISC CPU has 64 bit virtual address and 8 GB of RAM uses an inverted page table with 8 kB page size. What is the size of TLB ? ***(R.G.P.V., Dec. 2008)***

Ans. **Inverted Paging** – Refer Q.55 (iii).

TLB Size – With a 8 K page, each process needs $(2^{64-13}) = 2^{51}$ entries in its conventional page table. However, the number of entries in inverted page table is equal to the number of page frames in physical memory. Thus, the number of page frames in physical memory is 2^{20} $(2^{33}/2^{13})$. Thus, the size of TLB is 2^{20} entries.

Q.61. What is meant by page faults ? Will increase in degree of multiprogramming decrease page fault ? Justify your answer. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. An interrupt caused by program needing a specific page not yet in memory is referred to as ***page fault.***

As the multiprogramming level increases from a small value, one would expect to see processor utilization rise, because there is less chance that all resident processes are blocked. However, a point is reached at which the average resident set is inadequate. At this point, the number of page faults rises dramatically and processor utilization collapses.

Q.62. What are the circumstances under which page faults occur ? Describe the actions taken by the operating system when a page fault occurs. ***(R.G.P.V., June 2005)***

Or

How is page fault handled by operating system ? ***(R.G.P.V., Dec. 2005)***

Or

Under what circumstances do page fault occur ? Describe the action taken by the operating system when a page fault occurs. ***(R.G.P.V., June 2006)***

Ans. If the process tries to access a page that was not into memory or marked false (or invalid), causes a ***page fault trap***. The paging hardware, in translating the address through the page table, will notice that the invalid is set causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory, rather than an invalid address error as a result of an attempt to use an illegal memory address. We must therefore correct this oversight. The procedure for handling the page fault is as follows –

(i) An internal table is checked everytime for the process, to determine whether the reference was a valid or invalid memory access.

(ii) If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

(iii) By taking one from the free frame list we find a free frame.

(iv) We schedule a disk operation to read the desired page into the newly allocated frame.

(v) When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

(vi) We restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory.

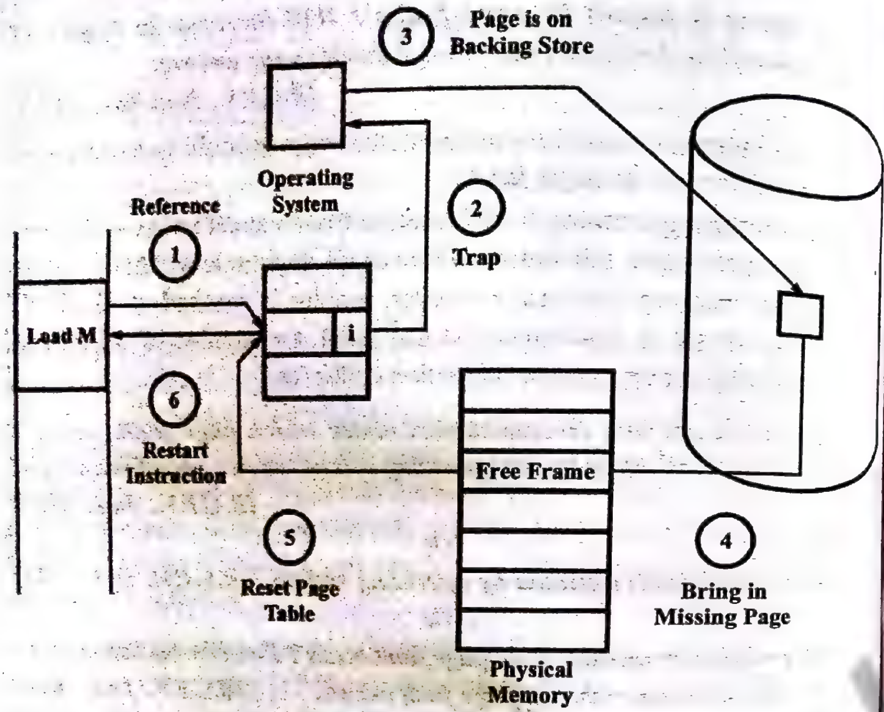


Fig. 3.38 Steps in Handling a Page Fault

It is important to realize that, because we save the state of the interrupted process when the page fault occurs, we can restart the process in exactly the same place, except that the desired page is now in memory and is accessible. In this way, we are able to execute a process, even though portions of it are not in memory.

Q.63. Explain the following –

(i) MFT ***(ii) Paging***

(iii) Segmentation ***(iv) TLB hit/miss.***

(R.G.P.V., Dec. 2012)

Ans. (i) MFT – Refer Q.41 (ii).

(ii) Paging – Refer Q.52.

(iii) Segmentaion – Refer Q.50.

(iv) TLB hit/miss – Refer Q.58.

Q.64. What is the page replacement technique ? Why it is needed ?

Or

Write the necessary step taken by the operating system when a page fault occurs.

(R.G.P.V., June 2016)

Ans. When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in.

If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date. However, if the page has not been changed, the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted.

Page replacement takes the following approach. If no frame is free, we find one that is not currently being used and free it. We can free a frame by writing its contents to swap space, and changing the page table to indicate that page is no longer in memory (fig. 3.39). The steps of handling the page fault with page replacement is given as follows –

(i) Find the location of the desired page on the disk.

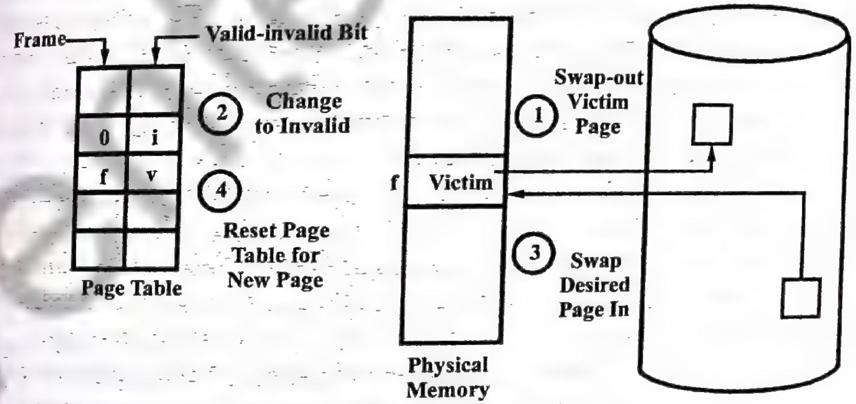


Fig. 3.39 Page Replacement

(ii) Find a free frame –

(a) If there is a free frame, use it.

(b) If there is no free frame, use a page-replacement algorithm to select a victim frame.

(c) Write the victim page to the disk, change the page and frame tables accordingly.

(iii) Read the desired page into the newly free frame, change the page and frame tables.

(iv) Restart the user process.

If no frames are free, two page transfers are required. This doubles the page fault service time and increases the effective access time accordingly.

This overhead can be reduced by using a modify bit. Each page or frame may have a modify bit associated with it, in the hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified. When we select a page for replacement we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the disk, thus we must write that

page to the disk. If the modify is not set, however the page has not been modified since it was read into memory. Therefore, if the copy of the page on the disk has not been overwritten, then we can avoid writing the memory page to the disk. This technique also applies to read-only pages. Such pages cannot be modified, thus, they may be discarded when desired. This scheme can reduce significantly the time required to service a page fault, since it reduces I/O time by one half if the page is not modified.

Q.65. What is the difference between global and local page replacement policy ? ***(R.G.P.V., June 2016)***

Ans. The page replacement algorithms can be divided into two categories – global replacement and local replacement. Global replacement allows a process to select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; one process can take a frame from another. Local replacement requires that each process select from only its own set of allocated frames.

For example, consider an allocation scheme where we allow high-priority processes to select frames from low-priority processes for replacement. A process can select a replacement from among its own frames or the frames of any lower-priority process. This approach allows a high-priority process to increase its frame allocation at the expense of the low-priority process.

With a local replacement strategy, the number of frames allocated to a process does not change. With global replacement, a process may happen to select only frames allocated to other processes, thus increasing the number of frames allocated to it.

The problem associated with a global replacement algorithm is that a process cannot control its own page-fault rate. The set of pages in memory for a process depends not only on the paging behaviour of that process, but also on the paging behaviour of other processes. Therefore, the same process may perform quite differently due to totally external circumstances. This is not the case with a local replacement algorithm. In this, the set of pages in memory for a process is affected by the paging behaviour of only that process. Thus, local replacement hinders a process by not making available to it, less used pages of memory. Therefore, global page replacement results in greater system throughput, and is more commonly used.

Q.66. Explain the concept of dirty bit for improving the performance during page fault. ***(R.G.P.V., Dec. 2013)***

Ans. Before overwriting a page in the memory, the O/S has to verify if that page has been modified after it was loaded from the disk. This type of modified page is known as dirty page. In general, these days the compilers generate reentrant code which does not modify itself. Therefore, the pages

reserved for the code area of a program generally never get dirty, but the ones for the data area can. The O/S preserves 1 bit for each physical page frame to show whether a page has become dirty or not. This bit is known as dirty bit.

The hardware is generally implemented in such a way that if a page is modified, the dirty bit for that related page frame is set to 1 automatically, otherwise it is set to 0. This support from the hardware is expected by the O/S supporting the virtual memory scheme. The hardware understands the write instruction utilised for updating a page from the opcode bits in the IR. It also understands the page frame modified by the resultant address derived after the address translation. In turn, the hardware can maintain this bit related to the modified page to 1. This type of bit is known as dirty bit because it shows whether a page is modified or not, and thus, whether it requires to be saved on the disk before getting overwritten.

If a page is dirty or modified, it should not be blindly overwritten, because otherwise, the updates will be lost. Hence, in this condition, this page has to be written back onto the disk at a suitable location, so that when it is brought in next time, the right and current copy is brought in. If the page is not modified, the O/S need not write it back. It is simply overwritten. In this manner, a lot of time is saved.

Q.67. Describe various page replacement algorithms along with their merits and demerits. ***(R.G.P.V., June 2008)***

Or

Describe the working of page replacement algorithm. ***(R.G.P.V., Dec. 2014)***

Or

Discuss any one page replacement algorithm. ***(R.G.P.V., June 2017)***

Ans. The various page replacement algorithms are explained as below –

(i) First-In-First-Out (FIFO) – The simplest page replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. A FIFO queue is maintain to hold all pages in memory. The pages are replaced at the head of the queue. When a page is brought into memory, it inserts at the tail of the queue.

For example, for the string in fig. 3.40, there are three frames which are initially empty. The first three references (7, 0, 1) cause page faults, and are brought into these empty frames. The next reference (2) replaces page 7, because page 7 was brought in first. Since 0 is the next reference and 0 is already in memory, so there is no fault for this reference. The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1, 2) to be brought in. Now, the next reference to 0 will fault. Page 1 is then replaced by page 0. This process continues as shown in fig. 3.40. There are 15 faults altogether.

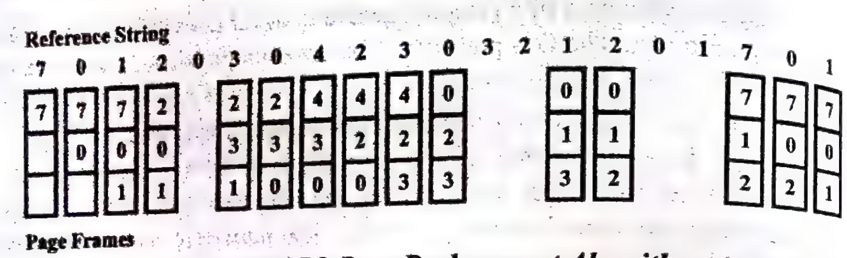


Fig. 3.40 FIFO Page Replacement Algorithm

The FIFO page replacement algorithm is easy to understand and program. However, its performance is not always good. The page replaced may be an initialization module that was used a long time ago and is no longer needed. On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

This algorithm also suffers from Belady's anomaly. We will study it.

Normally, if the physical memory size and the number of page frames available increase, the number of page faults should decrease. Due to this the performance should be enhanced. But this is not essentially the case if FIFO is used as the page replacement policy. Generally, this is referred to as Belady's anomaly. Let us, for example, take a reference string 2, 3, 4, 5, 2, 3, 6, 2, 3, 4, 5, 6.

Fig. 3.41 depicts that with 3 page frames, FIFO gives 9 page faults. Whereas, the number of page faults increases when we increase the number of page frames from 3 to 4 and becomes 10! This is clearly anomolous! and we can see it in fig. 3.41.

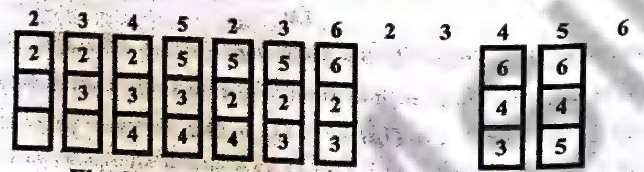


Fig. 3.41 Page Faults with 3 Page Frames

If we thoroughly see these figures, we can notice that the page faults increase because in the case depicted by fig. 3.42, the page is referenced as soon as it is evicted in this specific string. This type of anomolous behaviour can be rarely found for specific types of page reference strings. It does not always happen for all reference strings.

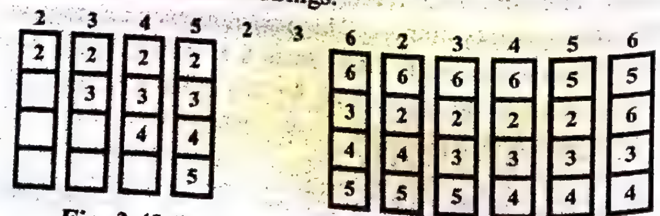


Fig. 3.42 10 Page Faults with 4 Page Frames

(ii) Optimal Page Replacement Algorithm – The optimal policy selects for replacement that page for which the time to the next reference is the longest. This algorithm has the lowest page-fault rate of all algorithms, and will never suffer from Belady's anomaly. It is also called OPT or MIN algorithm.

To understand the algorithm consider the following reference string –

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

The above string would yield nine page faults, as shown in fig. 3.43. The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because 7 will not be used until reference 18, whereas, page 0 will be used at 5, and page 1 at 14. The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again. With only nine page faults, optimal replacement is much better than a FIFO algorithm, which had 15 faults. In fact, no replacement algorithm can process this reference string in three frames with less than nine faults.

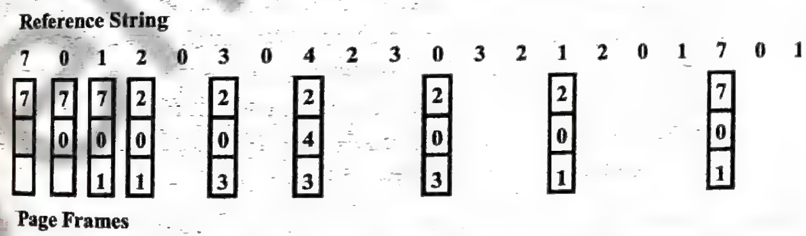


Fig. 3.43 Optimal Page Replacement Algorithm

Clearly, this algorithm is impossible to implement, because it would require the operating system to have perfect knowledge of future events. However, it does serve as a standard against which to judge other algorithms.

(iii) Least Recently Used (LRU) – The LRU policy is often used as a page-replacement algorithm and is considered to be good. When a page fault occurs, throw out the page that has been unused for the longest time. This approach is the least recently used (LRU) algorithm. In general, the LRU algorithm performs better than FIFO. The reason is that LRU takes into account the patterns of program behaviour by assuming that the page used in the most distant past is least likely to be referenced in the near future. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time. This strategy is the optimal page-replacement algorithm looking backward in time, rather than forward.

The result of applying LRU replacement to a reference string is shown in fig. 3.44. The LRU algorithm produces 12 faults. The first five faults are the same as the optimal replacement. When the reference to page 4 occurs, however, LRU replacement sees that, of the three frames in memory, page 2 we used least

recently. The most recently used page is page 0, and just before that page 3 was used. Thus, the LRU algorithm replaces page 2 without bothering that page 2 is about to be used. When it then faults for page 2, the LRU algorithm replaces page 3 since, of the three pages in memory {0, 3, 4}, page 3 is the least recently used. This process continues as shown in fig. 3.44. There are 12 faults altogether.

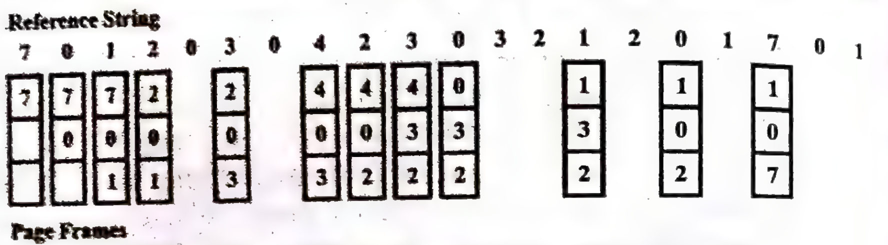


Fig. 3.44 LRU Page Replacement Algorithm

Although LRU is theoretically realizable, it is not cheap. To fully employ LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The difficulty is that the list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front is a very time consuming operation. Either (expensive) special hardware is needed, or to find a cheaper approximation in software.

Q.68. Explain Belady's algorithm. ***(R.G.P.V., Dec. 2013)***

Ans. Refer Q.67 (i).

Q.69. What do you mean by effective access time ?

Ans. Suppose p be the probability of a page fault ($0 \le p \le 1$). We would expect p to be close to zero, i.e., there will be only a few page faults. Then, the effective access time is

$$\text{EAT} = (1 - p) \times \text{ma} + p \times \text{Page fault time}$$

Effective access time is directly proportional to the paga fault rate. To compute the effective access time, we must know how much time is needed to service a page fault. For example, let average page fault service time is 20 miliseconds and memory access time 75 nanoseconds, then the effective access time in ns is

$$\begin{aligned}\text{EAT} &= (1 - p) \times 75 + p \times 20 \times 10^6 \\ &= 75 - 75p + 20{,}000{,}000\ p \\ &= 75 + 19999925\phi\end{aligned}$$

If one access out of 1000 causes a page fault, the effective access time is

$$\begin{aligned}\text{EAT} &= 75 + 19999925 \times 1 \\ &= 20{,}000{,}000 \\ &= 20 \text{ microseconds}\end{aligned}$$

Q.70. Explain the term locality of reference. ***(R.G.P.V., Dec. 2011)***

Ans. Some portions of program may not be referenced and therefore need not be loaded on a particular run. Unfortunately, in general we cannot predict the identity of superfluous pages or segments before the program is actually executed. In other words, the fact that some portions of the address space are superfluous on some runs mostly provides the justification for execution of partially loaded programs but offers little guidance as to which pages to load in order to reduce the number of page faults. Several studies have suggests a strong tendency of programs to favor subsets of their address space during execution. This phenomenon is known as ***locality of reference***. Both temporal and spatial locality of reference have been observed. Spatial locality is the tendency for a program to reference clustered locations in preference to randomly distributed locations. Temporal locality is the tendency for a program to reference the same location or a cluster several times during brief intervals of time.

Temporal locality of reference is exhibited by program loops, where a certain set of instructions and data are repetitively referenced for a period of time. Spatial locality suggests that once an item is referenced, there is high probability that it or its neighbouring items are going to be referenced in the near future. A locality is a small cluster of not necessarily adjacent pages to which most memory references are made during a period of time. Both temporal and spatial locality of reference are dynamic properties in the sense that the identity of the particular pages that compose the actively used set varies with time. The probability that a particular memory reference is going to be made to specific page is a time-varying function. This increases when pages in its current locality are being referenced, and it decreases otherwise.

From the virtual memory manager's point of view, locality of reference basically suggests that a significant portion of memory references of running process may be made to subset of its pages. Also there is increased probability that recently referenced pages, pages in the same locality, are going to be referenced in the near future. These findings may be utilized for implementation of replacement and allocation policies.

Q.71. What is working-set model ? Why it is used ?

Or

Discuss the working set model. ***(R.G.P.V., Dec. 2008)***

Ans. The working-set model is based on the assumption of locality. This model uses a parameter, Δ, to define the ***working-set window***. The idea is to examine the most recent Δ page references. The set of pages in the most recent Δ page reference is the working-set as shown in fig. 3.45. If a page is in active use, it will be in the working-set. If it is no longer being used, it will

drop from the working-set Δ time units after its last reference. Thus, the working-set is an approximation of the program's locality.

For example, the sequence of memory references shown in fig. 3.45, if $\Delta = 10$ memory references, then the working-set at time t_1 is {1, 2, 5, 6, 7}. By time t_2, the working-set has changed to {3, 4}.

The accuracy of the working-set depends on the selection of Δ. If Δ is too small, it will not encompass the entire locality. If Δ is too large, it may overlap several localities. If Δ is infinite, the working-set is the set of pages touched during the process execution.

The most important property of working-set is its size. If the working-set size is WSS_i for each process in the system, then

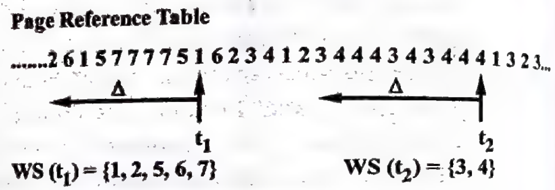


Fig. 3.45 Working-set Model

$$D = \Sigma WSS_i$$

where D is the total demand for frames. Each process is actively using the pages in its working-set. Thus, process i needs WSS_i frames. If the total demand is greater than the total number of available frames, thrashing will occur.

The use of working-set model is simple. The operating system monitors the set of each process and allocates to that working-set enough frames to provide it with its working-set size. If there are enough extra frames, another process can be initiated. If the sum of the working-set sizes increases, exceeding the total number of available frames, the operating system selects a process to suspend. The process' pages are written out and its frames are reallocated to other processes. The suspended process can be started later. Thus, this working-set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible and optimizes CPU utilization. The difficulty with the working-set model is keeping track of the working-set because the working-set window is a moving window.

Q.72. What is locality of reference and explain its use ? What is working set ? What is it used for ? (R.G.P.V., June 2017)

Ans. Locality of Reference – Refer Q.70.

Use – (i) Virtual memory management.

(ii) Paging.

Working Set – Refer Q.71.

Use – The working-sets are used in paging to ensure virtual memory management.

Q.73. What are the two major differences between segmentation and paging ?

Or

Compare paging and segmentation. (R.G.P.V., Dec. 2015)

Or

Differentiate between paging and segmentation. (R.G.P.V., Dec. 2016)

Ans. The two major differences between segmentation and paging are as follows –

(i) In segmentation, the program is divided into variable-size segments. In paging, the program is divided into fixed-size pages.

(ii) In segmentation, the user is responsible for dividing the program into segments. In paging, the division into pages is performed by the operating system and is transparent to the user.

Q.74. Why are segmentation and paging sometimes combined into one scheme ? (R.G.P.V., June 2007)

Or

Why paging and segmentation is combined ? (R.G.P.V., Dec. 2014)

Ans. Both segmented and paged implementation of virtual memory have their respective advantages and disadvantages, and neither is superior to the other over all characteristics. Some computer systems combine the two approaches in order to enjoy the benefits of both. One popular approach is to use segmentation from the user's point of view but to divide each segment into pages of fixed size for purposes of allocation. In this way, the combined system retains most of the advantages of segmentation. At the same time, the problems of complex segment placement and management of secondary memory are eliminated by using paging.

Q.75. Describe the segmented paging scheme of memory management and the hardware required to support the system. (R.G.P.V., Dec. 2010)

Or

Explain paged segmentation with its hardware implementation. (R.G.P.V., June 2009)

Ans. If the segments are large, it may be inconvenient or impossible, to keep them in main memory. This leads to the idea of paging them, so that only those pages that are actually needed have to be around. This combination is best illustrated by the architecture of the Intel 386. The 386 uses segmentation with paging for memory management. The maximum number of segments per process is 16 KB, and each segments can be as large as 4 GB. The page size is 4 KB.

The logical address space of a process is divided into two partitions. The first partition consists of up to 8 KB segments that are private to that process. The second partition consists of up to 8 KB segments that are shared among all the processes. Information about the first partition is kept in the *local*

descriptor table (LDT), information about the second partition is kept in the ***global descriptor table (GDT).*** Each entry in the LDT and GDT consists of 8 bytes, with detailed information about a particular segment including the base location and length of that segment.

The logical address is a pair (selector, offset), where the selector is a 16-bit number –

s	g	p
13	1	2

in which s designates the segment number, g indicates whether the segment is in the GDT or LDT, and p deals with protection. The offset is a 32-bit number specifying the location of the byte (or word) within the segment.

The machine has six segment registers allowing six segments to be addressed at any one time by a process. It has six 8 byte microprogram registers to hold the corresponding descriptors from either the LDT or GDT.

The physical address on the 386 is 32 bits long and is formed as follows - The segment register points to the appropriate entry in the LDT or GDT. The base and limit information about the segment are used to generate a linear address. First, the limit is used to check for address validity. If the address is not valid, a memory fault is generated, resulting in a trap to the operating system. If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address. This address is then translated into a physical address.

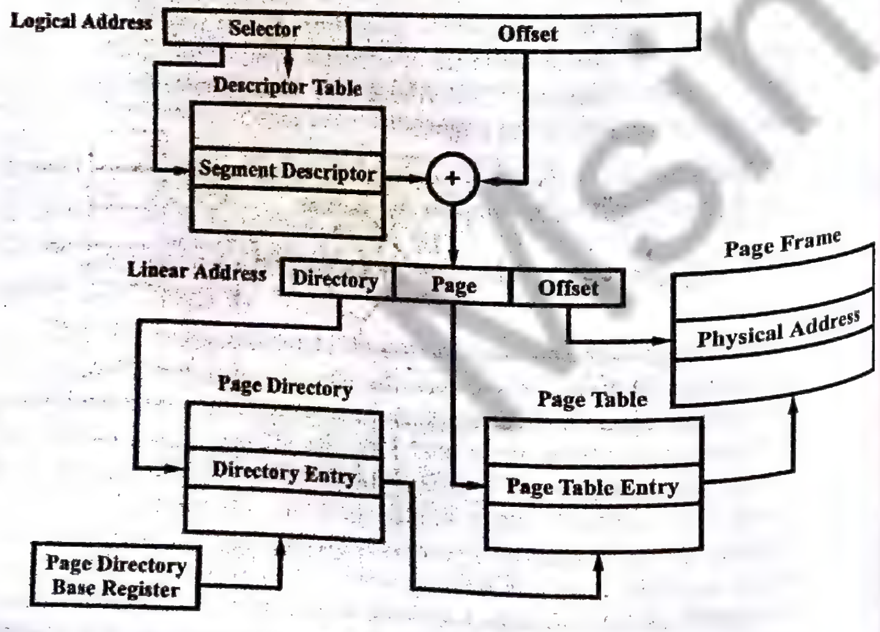


Fig. 3.46 Intel 80386 Address Translation

Since, each segment is paged and each page is 4 KB, a page table may thus consists of upto 1 million entries. Because each entry consists of 4 bytes, each process may need upto 4 MB of physical address space for the page table. It is clear that we do not want to allocate the page table contiguously in main memory. The solution adopted in the 386 is to use a two-level paging scheme. The linear address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits. Since we page the page table, the page number is further divided into a 10-bit page directory pointer and a 10-bit page table pointer. The logical address is as follows –

Page number		Page offset
p_1	p_2	d
10	10	12

The address translation scheme for this architecture is shown in fig. 3.46 To improve the efficiency of physical memory use, Intel 386 page tables can be swapped to disk. In this case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is in memory or disk. If the table is on disk, the operating system can use the other 31 bits to specify the disk location of the table; the table then can be brought into memory on demand.

Q.76. Explain how logical memory address are translated into physical memory address in segmented memory management system. ***(R.G.P.V., June 2017)***

Ans. In translation of local memory address to physical memory address, segment map table (SMT) or segment descriptor table (SDT) is used. Let us assume an SMT created by OS is shown in fig. 3.47.

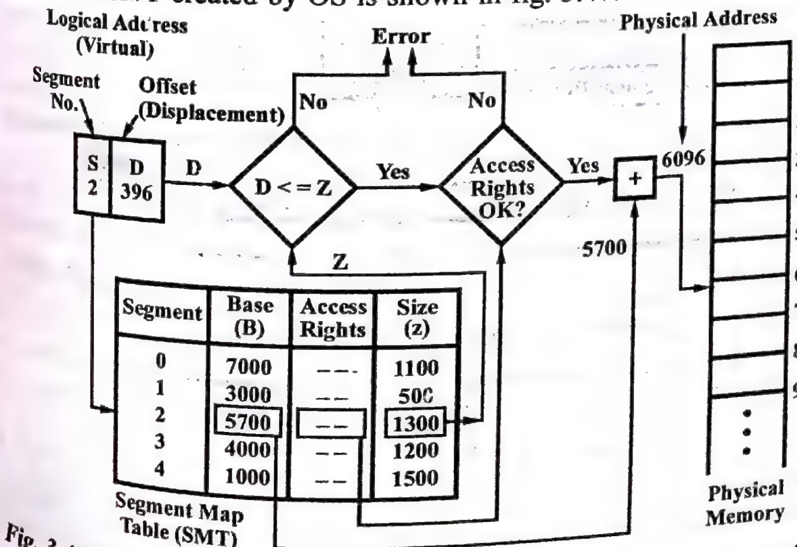


Fig. 3.47 Address translation in Segmented Memory Management System

Let us assume that virtual address to be translated is – 1996 clearly in SMT, segment 0 have size 1100 and segment 1 have size 500 and sum of both segment's size is less than 1996 but when we add the size of segment 0 the total sum will be greater than 1996. It means 1996 lies in segment 2 so we have S = 2 and

D = Virtual address – (Size of segment 0 + size of segment 1)

D = 1996 – (1100 + 500)

D = 396

So, S = 2 and D = 396 is logical address.

Steps to transfer logical address into physical address are as follows –

(i) Logical address is initially saved in instruction register from which the higher order bit is taken out that represent the segment number S i.e. S = 2. It will be the input to the SMT.

(ii) The OS stores the data of row from SMT whose index number is S (i.e., 2). Hence, it will store

Z = 1300 and Base address, B = 5700 in memory.

(iii) Since, D is the displacement address so it must be less than Z (i.e. the size of segment). Hence, compare the D with Z to ensure D <= Z. If its not, then the error will occour.

(iv) If the displacement is legal then OS checks whether the access rights to that memory are given to current user or not. If not then error will occur otherwise continue.

(v) Now calculate the effective address as B + D this will be 5700 + 396 = 6096.

(vi) This address is required physical address which maps in physical memory.

NUMERICAL PROBLEMS

Prob.10. Consider the logical address space of 8 pages of 1024 words mapped onto physical memory of 32 frames –

(i) How many bits are there in logical address ?

(ii) How many bits are there in physical address ?

(R.G.P.V., June 2007, 2009)

Sol. Given the page size = 1024 words = 1 K.

Since, the page size and frame size are equal in paging system.

Therefore,

Logical-address space = 8 K = $2^3 \times 2^{10} = 2^{13}$ bytes

and physical-address space = 32 K = 2^{15} bytes

The size of logical-address space is 2^{13}, and a page size is 2^{10} bytes, then the high order (13 – 10) bits of the logical address designate the page

number, and the 10 low-order bits of the logical address designate the page offset.

(i) There are 13 bits in the logical address

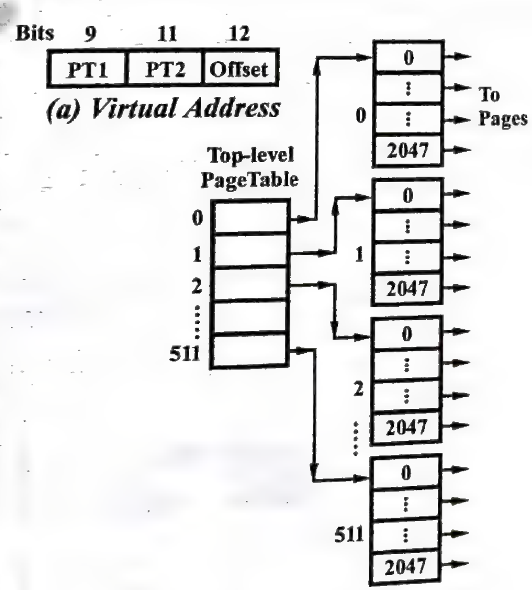
(ii) There are 15 bits in physical address.

Prob.11. A Computer with 32-bits virtual address space uses two level page table. Virtual addresses are split into 9 bit top level and 11 bit second level page table field and an offset. Describe how virtual address will be translated to physical address with neat sketch ? Physical address is of 20 bits.

(R.G.P.V., Dec. 2008)

Sol. In fig. 3.48 (a), we have a 32-bit virtual address that is partitioned into a 9-bit PT1 field, a 11-bit PT2 field and a 12-bit offset field. Since offsets are 12 bits, pages are 4K and there are a total of 2^{20} of them.

Essentially, the big page map table with 2^{20} entries are broken up into 2^9 or 512 small page map table (0-511) with each PMT having 2^{11} or 2048 (0-2047) entries. PT1, which has 9 bits is used as an index into the top-level of page table to choose the appropriate small PMT. PT2, which has 11 bits then is used as an index to select appropriate entry within that PMT. That entry will give the page frame number (F) corresponding to virtual page number (PT1 + PT2). The offset can now be concatenated to this page frame number (F) to form the final resultant physical address.



(a) Virtual Address

(b) Two-level Page Table

Fig. 3.48

Prob.12. Assume memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order). How would each of first fit, best fit and worst fit algorithms place processes of 212 KB, 417 KB, 112 KB and 426 KB (in order) ? Which algorithm makes the most efficient use of memory ?

(R.G.P.V., June 2004, 2008, 2009)

Sol. The physical memory with given memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order) is as shown in fig. 3.49.

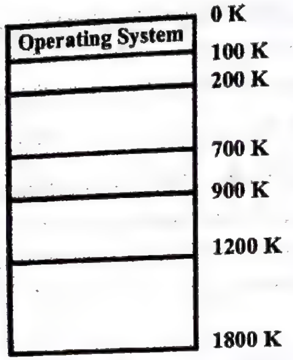


Fig. 3.49

(i) First Fit Strategy – In this strategy, the first hole that is big enough, is allocated. First the process of 212 KB is allocated in the hole of 500 KB, resulting in free space of 288 KB. Now, the request for process of 417 KB is met in the hole of 600 KB, resulting in free space of 183 KB. Now, the request for process of 112 KB is met in the hole of 200 KB, resulting in the free space of 88 KB. Now, the request for process of 426 KB cannot be met. However, the total free space of 559 KB is available due to internal fragmentation (fig. 3.50).

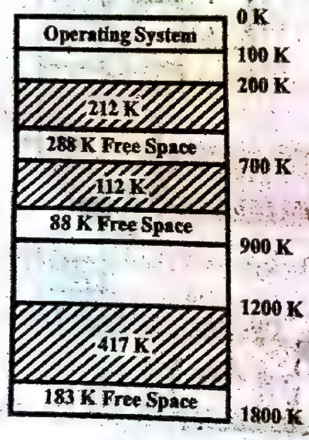


Fig. 3.50

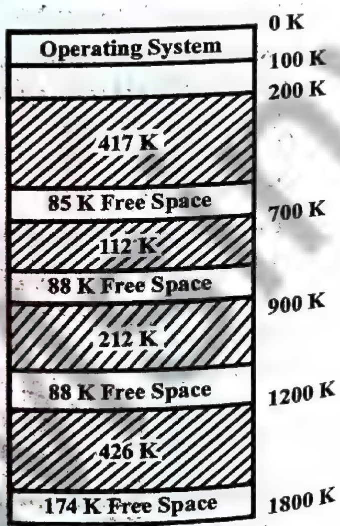


Fig. 3.51

(ii) Best Fit Strategy – In this strategy, the block that is closest in size to the request is selected for allocation. So, the first process of 212 KB is placed in hole of 300 KB, resulting in free space of 88 KB. Now, the second process of 417 KB is placed in hole of 500 KB, resulting in free space of 83 KB. Now, the third process of 112 KB is placed in hole of 200 KB, resulting in free space of 88 KB. At last, the fourth process of 426 KB is also placed in

hole of 600 KB, resulting in free space of 174 KB (fig. 3.49). Here, the total free space is 433 KB due to internal fragmentation. Thus, all the four processes's request are met and less internal fragmentation.

(iii) Worst Fit Strategy – In this strategy, we allocate the largest hole instead of tiny hole which may not be used, that is always take the largest available hole so that hole broken off will be enough to be useful.

In fig. 3.52 requests for memory of 212 KB, 417 KB and 112 KB have been satisfied. But, the request for memory of size 426 KB has not been satisfied. However, there is sufficient free space to satisfy this request in main memory.

Best fit algorithm contain less fragmentation than others. However, it is slow than first fit because it must search the entire list every time it is called. In our case the Best fit algorithm is efficient, because searching needed less time than compaction of whole memory.

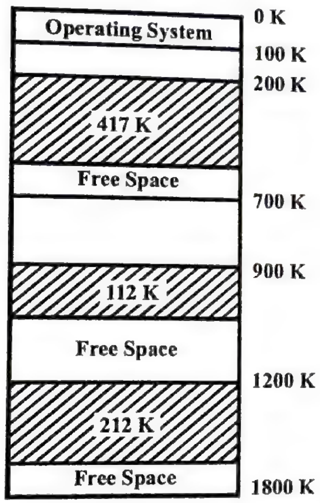


Fig. 3.52

Prob.13. Consider variable partitioning. If the following jobs in order are to fit in memory 200 K, 50 K, 250 K, 150 K using –

(i) First fit

(ii) Best fit

(iii) Worst fit.

With the neat sketches explain how these jobs will be placed. Present memory position is (fig. 3.53) –

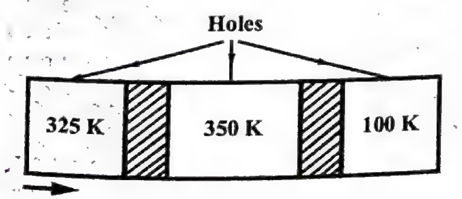


Fig. 3.53

(R.G.P.V., Dec. 2008)

Sol. *(i) First Fit* – First fit searches for the first hole larger than or equal to 200 K as per requirement. So that first hole 325 K can accommodate the request. So the first request is allocated to the first hole. This would reduce the size of the first hole to 125 K as shown in fig. 3.54 (a).

Then after searches from the first hole for larger than or equal to 50 K as per requirement. So that first reduced hole 125 K can accommodate the request and allocated to the hole 125 K, So that would reduce the size of hole to 75 K as shown in fig. 3.54 (b).

Then third request 250 K would be allocated into the second largest hole 350 K. Hole would be reduce to 100 K as shown in fig. 3.54 (c).

The final request, 150 K must wait.

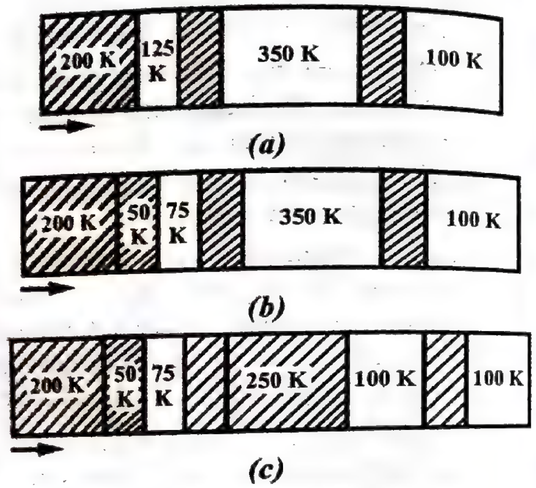


Fig. 3.54

(ii) Best Fit –

Best fit searches for the request 200 K, a suitable hole larger than or equal to 200 K. The first hole is 325 K making it the best fit. So that the request can be accommodated and reduces the size of hole to 125 K as shown in fig. 3.55 (a).

For the request 50 K, third hole is larger than 50 K, so that we can accommodate this request on the third hole and reduces the size of hole to 50 K as shown in fig. 3.55 (b).

For the request 250 K, second hole is suitable and full-fill the requirement of this request we can accommodate this request, so reduce the size of first hole to 100 K as shown in fig. 3.55 (c).

At last for 150 K there is no appropriate space hole to accommodate this request, even though the sum of all the holes is larger than the storage needed by the request.

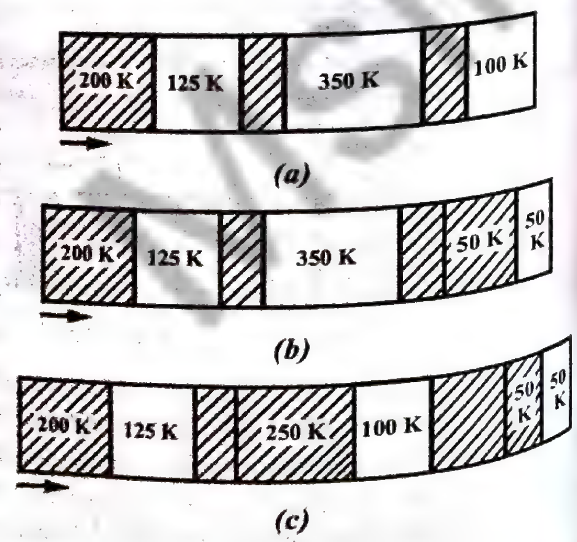


Fig. 3.55

(iii) Worst Fit – Worst fit always allocate large hole. So the first request of 200 K would be allocated to the second hole of 350 K and reduce the size of hole to 150 K as shown in fig. 3.56 (a).

For the second request 50 K we would accommodate remaining highest node. So we will allocate this request to 325 K hole and hole would be reduced to 275 K as shown in fig. 3.56 (b).

For the request 250 K, remaining highest hole is 275 K. So we would allocate this request to first hole and hole would be reduced to 25 K as shown in fig. 3.56 (c).

For the request 150 K remaining highest hole is 150 K. So we would allocate to second hole as shown in fig. 3.56 (d).

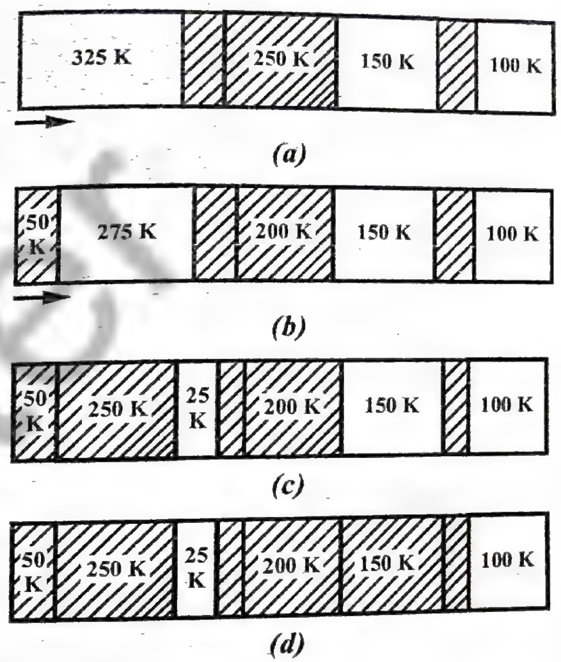


Fig. 3.56

Prob.14. Consider a swapping system in which memory consists of the following hole sizes in memory order 10 KB, 4 KB, 20 KB, 18 KB, 7 KB, 9 KB, 12 KB and 15 KB. Which hole is taken for successive segment requests of –

(i) 12 KB (ii) 10 KB (iii) 9 KB

for first fit and best fit.

(R.G.P.V., June 2010)

Sol. Similar as Prob.13.

Prob.15. Suppose a paging system has 2^{g+h} virtual addresses and uses 2^{h+k} locations in primary memory for integers g, h and k. What is the page size of the system that is implied by the virtual and physical address sizes ? How many bits are required to store a virtual address ? (R.G.P.V., June 2008)

Sol. Virtual Address Space = 2^{g+h} bytes

Physical Memory = 2^{h+k} bytes

Since page size and frame size are equal, hence, if page size is 2^h then number of frames is 2^k.

The number of bits required to store a virtual address is g + h.

Prob.16. Consider a paging system with the page table stored in memory.

(i) If a memory reference takes 1.2 microseconds, how long does a paged memory reference take ?

(ii) If we add 8 associative registers and 75% of all page table reference are found in the associative registers, what is the effective memory reference time ?

(Assume that finding a page table entry in the associative registers takes zero time, if it is there). ***(R.G.P.V., June 2009)***

Sol. (i) Time to access memory for the page table and frame number = 1.2 μsec

Time to access the desired byte in memory = 1.2 μsec

Total time = 2.4 μsec.

(ii) Effective memory access time

$$= 0.75 \times 1.2 + 0.25 \times 2.4$$
$$= 0.9 + 0.6 = 1.5 \ \mu\text{sec}$$

Prob.17. Consider a paged segmentation scheme. The virtual address is of 32-bit has 12-bits for segment number. The size of page is 4096 words and physical memory is of 22 bits. With neat sketch, explain how virtual address is mapped with physical address. ***(R.G.P.V., Dec. 2008)***

Sol. A virtual address in a paged segmentation scheme consists of two parts – the segment and the address within the segment. The address within the segment is further divided into a page number and a word within the page as shown in fig. 3.57.

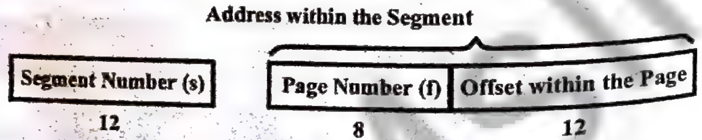


Fig. 3.57 A 32-bit Virtual Address

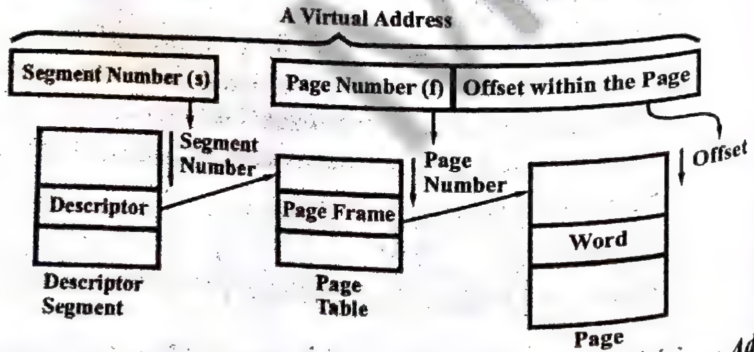


Fig. 3.58 Conversion of a Two-part Address into a Main Memory Address

Physical memory is of 22-bits and the page size is 2^{12} words. Therefore, physical memory is divided into 2^{10} ($2^{22}/2^{12}$) page frames.

When a memory reference occurs, a register (the descriptor base register), is used to locate the descriptor segment's page table, which in turn, points to the pages of the descriptor segment. Once the descriptor for the needed segment has been found, the addressing proceeds as shown in fig. 3.58.

Prob.18. In a paged segmented system, a virtual address consists of 32 bits of which 12 bits are used for displacement, 11 bits are for segment number and 9 bits are for page number. Calculate –

(i) Page size ***(ii) Max. segment size***

(iii) Max. no. of pages ***(iv) Max. no. of segments.***

(R.G.P.V., Dec. 2010)

Sol. (i) Page Size –

Size, displacement = 12 bits

So, page size = 2^{12} **Ans.**

(ii) Max. Segment Size –

Since no. of bits for page number = 9

and no. of bits for page offset = 12

Therefore segment offset = 21 bits

Hence, max. segment size = 2^{21} **Ans.**

(iii) Max. No. of Pages –

Since, page number = 9 bits

Therefore, max. no. of pages = 2^9 **Ans.**

(iv) Max. No. of Segments –

Since, segment number = 11 bits

Therefore, max. no. of segments = 2^{11} **Ans.**

Prob.19. Consider the following page reference string –

1, 2, 3, 1, 4, 5, 6, 2, 1, 3, 2, 7, 6, 3, 4, 1, 2, 6.

How many page faults would occur for the LRU replacement algorithm assuming six frames ? All frames are initially empty. (R.G.P.V., June 2010)

Sol. The working of LRU page replacement algorithm for the given page reference string is shown in fig. 3.59. There are total 8 page faults.

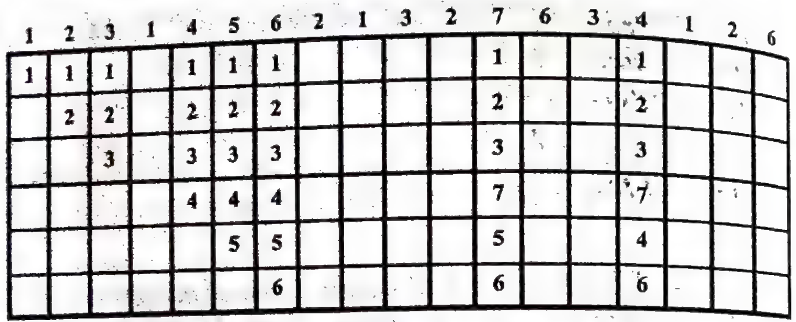


Fig. 3.59 LRU Page Replacement Using 6 Frames

Prob.20. Consider the main memory with capacity of 4 page frame. Assume that the pages of a process are referenced in the order as given below –

1, 3, 4, 4, 3, 2, 1, 7, 5, 6, 4, 2, 1, 2.

Which one is better FIFO or LRU and why ?

(R.G.P.V., Dec. 2013)

Sol. **FIFO** – The working of FIFO page replacement algorithm for the given page reference string is shown in fig. 3.60. There are 10 page faults.

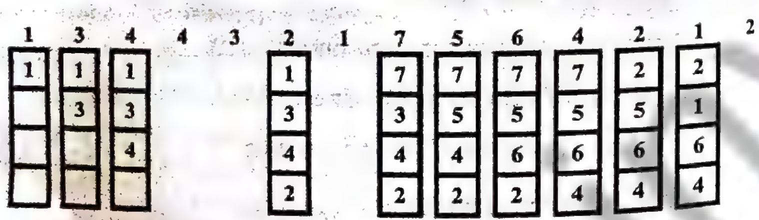


Fig. 3.60 FIFO Page Replacement Using 4 Frames

LRU – The working of LRU page replacement algorithm for the given page reference string is shown in fig. 3.61. There are 10 page faults.

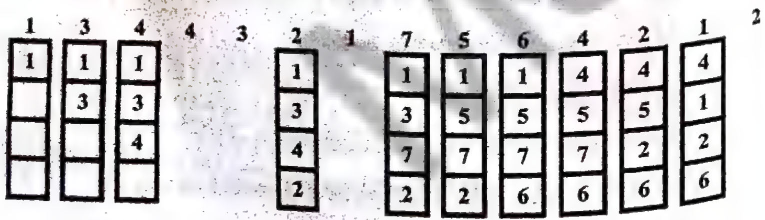


Fig. 3.61 LRU Page Replacement Using 4 Frames

When the number of page faults is equal then we use FIFO page replacement algorithm due to its simplicity. In that case, LRU is not considered due to the implementation overheads.

Prob.21. Prove the Belady's anomaly in FIFO page replacement algorithm, assuming that the number of frames is increased from 3 to 4. For the page reference strings

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Assume all frames are empty initially.

(R.G.P.V., May 2018)

Sol. The working of FIFO page replacement algorithm for the given page reference string is shown in fig. 3.62.

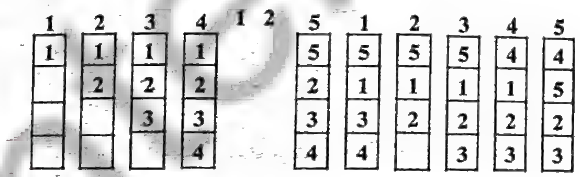


Fig. 3.62 FIFO Page Replacement using 4 Frames

Prob.22. Consider the following page reference string –

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9.

How many page faults would occur for the following replacement algorithms, assuming 2, 4 and 5 frames being made available ?

(i) FIFO (ii) LRU.

(R.G.P.V., June 2009)

Sol. (i) FIFO – The FIFO replacement process is shown in fig. 3.63. Given reference string is

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9

(a) 2-Frames – With two frames, there are 15 page faults [see fig. 3.63 (a)].

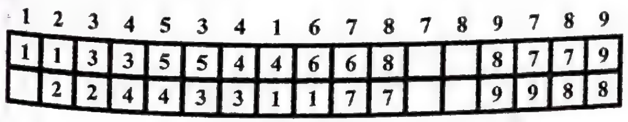


Fig. 3.63 (a)

(b) 4 Frames – With 4 frames, there are 10 page faults [see fig. 3.63 (b)].

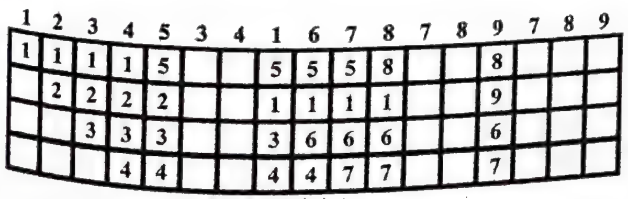


Fig. 3.63 (b)

(c) **5 Frames** – With 5 frames, there are 9 page faults [see fig. 3.63 (c)].

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9
1	1	1	1	1				6	6	6			6			
	2	2	2	2				2	7	7			7			
		3	3	3				3	3	8			8			
			4	4				4	4	4			9			
				5				5	5	5			5			

Fig. 3.63 (c)

(ii) *LRU* – The LRU replacement is shown in fig. 3.64.

(a) **2 Frames** – With 2 frames, there are 15 page faults. [see fig. 3.64 (a)].

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9
1	1	3	3	5	5	4	4	6	6	8			8	7	7	9
	2	2	4	4	3	3	1	1	7	7			9	9	8	8

Fig. 3.64 (a)

(b) **4 Frames** – With 4 frames, there are 10 page faults [see fig. 3.64 (b)].

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9
1	1	1	1	5			5	6	6	6			6			
	2	2	2	2			1	1	1	1			9			
		3	3	3			3	3	7	7			7			
			4	4			4	4	4	8			8			

Fig. 3.64 (b)

(c) **5 Frames** – With 5 frames, there are 9 page faults [see fig. 3.64 (c)].

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9
1	1	1	1	1				1	1	1			1			
	2	2	2	2				6	6	6			6			
		3	3	3				3	3	8			8			
			4	4				4	4	4			9			
				5				5	7	7			7			

Fig. 3.64 (c)

Prob.23. Consider the following page reference strings 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9.

How many page faults would occur for LRU replacement algorithm ? Assume frame size = 4. ***(R.G.P.V., Dec. 2014)***

Sol. Refer to Prob.22 (ii) (b).

Prob.24. How many page faults will occur with a reference string 0172327103? There are four frames which are initially empty. Use –
(i) FIFO (ii) LRU (iii) LFU (iv) Optimal.
Page replacement algorithms.

(R.G.P.V., Dec. 2008)

Sol. *(i)* *FIFO* – The working of FIFO page replacement algorithm for the given page reference string is as shown in fig. 3.65.

0	1	7	2	3	2	7	1	0	3
0	0	0	0	3				3	
	1	1	1	1				0	
		7	7	7				7	
			2	2				2	

Fig. 3.65 FIFO Page Replacement Using 4 Frames

Thus, there are total **6** page faults.

(ii) *LRU* – The working of LRU page replacement algorithm for the given page reference string is as shown in fig. 3.66.

0	1	7	2	3	2	7	1	0	3
0	0	0	0	3				0	0
	1	1	1	1				1	1
		7	7	7				7	7
			2	2				2	3

Fig. 3.66 LRU Page Replacement Using 4 Frames

Thus, there are total **7** page faults.

(iii) *LFU* – The working of LFU page replacement algorithm for the given page reference string is as shown in fig. 3.67.

0	1	7	2	3	2	7	1	0	3
0	0	0	0	3				0	3
	1	1	1	1				1	1
		7	7	7				7	7
			2	2				2	2

Fig. 3.67 LFU Page Replacement Using 4 Frames

Thus, there are total **7** page faults.

(iv) Optimal – The working of optimal page replacement algorithm for the given page reference string is as shown in fig. 3.68.

0	1	7	2	3	2	7	1	0	3
0	0	0	0	3				3	
	1	1	1	1				0	
		7	7	7				7	
			2	2				2	

Fig. 3.68 Optimal Page Replacement Using 4 frame

Thus, there are total 6 page faults.

Prob.25. Find the number of page faults with reference string 0172327103 having 4 page frames and 8 pages. All frames are initially empty. Page replacement strategies are –

(i) FIFO (ii) LRU (iii) Optimal. ***(R.G.P.V., Dec. 2010)***

Sol. Refer Prob.24.

Prob.26. Consider the following reference string –

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming three, five or six frames –

(i) LRU (ii) Optimal. ***(R.G.P.V., June 2011)***

Sol. (i) LRU – Given string is 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

(a) 3 Frames – The process is shown in fig. 3.69. There occur 15 page faults.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	4		4	5	5	5	1		1	7	7		2	2			2
	2	2	2		2	2	6	6	6		3	3	3		3	3			3
		3	3		1	1	1	2	2		2	2	6		6	1			6

Fig. 3.69 LRU Replacement with 3 Frames

(b) 5 Frames – This process causes 8 page faults. The process of LRU replacement is shown in fig. 3.70.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1				1	1							
	2	2	2			2	2				2	2							
		3	3			3	6				6	6							
			4			4	4				3	3							
						5	5				5	7							

Fig. 3.70 LRU Replacement with 5 Frames

(c) 6 Frames – The process is shown in fig. 3.71. There are 7 page faults.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1					1							
	2	2	2			2	2					2							
		3	3			3	3					3							
			4			4	4					7							
						5	5					5							
							6					6							

Fig. 3.71 LRU Replacement with 6 Frames

(ii) Optimal – Given string is 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

(a) 3 Frames – This will cause 11 page faults. This process is shown in fig. 3.72.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1				3	3			3	3			6
	2	2	2			2	2				2	7			2	2			2
		3	4			5	6				6	6			6	1			1

Fig. 3.72 Optimal Replacement with 3 Frames

(b) 5 Frames – This causes 7 page faults. This is shown in fig. 3.73.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1					1							
	2	2	2			2	2					2							
		3	3			3	3					3							
			4			4	6					6							
						5	5					7							

Fig. 3.73 Optimal Replacement with 5 Frames

(c) 6 Frames – This process also causes 7 page faults. This is shown in fig. 3.74.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1					1							
	2	2	2			2	2					2							
		3	3			3	3					3							
			4			4	4					7							
						5	5					5							
							6					6							

Fig. 3.74 Optimal Replacement with 6 Frames

Prob.27. Consider the following page reference string –

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults will occur for LRU, FIFO and optimal page replacement algorithms, assuming 4 available frames (initially all empty) ? ***(R.G.P.V., Dec. 2011)***

Sol. **LRU** – This process causes 10 page faults. The process of LRU replacement is shown in fig. 3.75.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1				1	1	6			6			
	2	2	2			2	2				2	2	2			2			
		3	3			5	5				3	3	3			3			
			4			4	6				6	7	7			1			

Fig. 3.75 LRU Replacement with 4 Frames

FIFO – This is shown in fig. 3.76. There are 14 page faults altogether.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			5	5	5	5		3	3	3		3	1		1	
	2	2	2			2	6	6	6		6	7	7		7	7		3	
		3	3			3	3	2	2		2	2	6		6	6		6	
			4			4	4	4	1		1	1	1		2	2		2	

Fig. 3.76 FIFO Replacement with 4 Frames

Optimal – The optimal policy with four frames gives 8 page faults. This process is shown in fig. 3.77.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	1	1	1			1	1					7				1			
	2	2	2			2	2					2				2			
		3	3			3	3					3				3			
			4			5	6					6				6			

Fig. 3.77 Optimal Replacement with 4 Frames

Prob.28. Consider the following page reference string –

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms for three and four frames ? Remember all frames are initially empty, so your first unique pages will all cost one fault each.

(i) LRU replacement

(ii) FIFO replacement

(iii) Optimal replacement.

(R.G.P.V., Dec. 2012, 2016)

Sol. (i) LRU Replacement –

(a) 3 Frames – Refer to Prob.26 (i) (a).

(b) 4 Frames – Refer to Prob.27 (Under the heading LRU).

(ii) FIFO Replacement –

(a) 3 Frames – This will cause 16 page faults. This process is shown in fig. 3.78.

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	1	3	6
1	1	1	4		4	4	6	6	6		3	3	3		2	2		2	6
	2	2	2		1	1	1	2	2		2	7	7		7	1		1	1
		3	3		3	5	5	5	1		1	1	6		6	6		3	3

Fig. 3.78 FIFO Replacement with 3 Frames

(b) 4 Frames – Refer to Prob.27 (Under the heading FIFO)

(iii) Optimal Replacement –

(a) 3 Frames – Refer to Prob.26 (ii) (a).

(b) 4 Frames – Refer to Prob.27 (Under the heading optimal).

Prob.29. Consider the following sequence of memory references from a 460 word program –

10, 11, 104, 170, 73, 309, 185, 245, 246, 434, 458 and 364.

Page size is of 100 words and 200 words of primary memory is available.

(i) Give the reference string

(ii) Calculate page fault rate by FIFO, LRU and optimal page replacement algorithm. ***(R.G.P.V., Dec. 2008)***

Sol. (i) Given the program size is 460 words or bytes. Also page size is of 100 words and 200 words of primary memory is available. We know that the physical memory is dived into frames and logical memory is divided into pages.

Both page and frame is of the same size. Thus, 460 words prorgam is divided into 5 pages 0, 1, 2, 3 and 4. And 200 words of memory is divided into 2 frames of 100 words each.

Therefore, the refrence string is –

0, 0, 1, 1, 0, 3, 1, 2, 2, 4, 4, 3.

(ii) FIFO – With 2 frames, there are 6 page faults.

0	0	1	1	0	3	1	2	2	4	4	3
0		0			3		3		4		4
		1			1		2		2		3

Fig. 3.79 FIFO Replacement with 2 Frames

LRU – With 2 frames, there are 7 page faults.

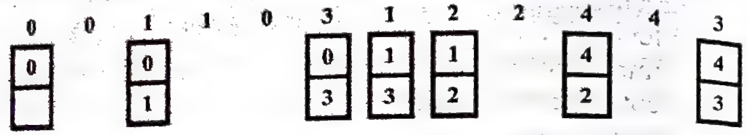


Fig. 3.80 LRU Replacement with 2 Frames

Optimal – With 2 frames, there are 5 page faults.

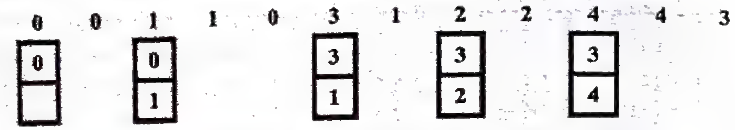


Fig. 3.81 Optimal Replacement with 2 Frames

Prob.30. If the average page fault service time of 25 ms and a memory access time of 100 ns. Calculate the effective access time. ***(R.G.P.V., May 2018)***

Sol. Here given,

Page fault service time = 25 ms

Memory access time (Ma) = 100 ns.

We know, effective access time (EAT) = (1 – P) × ma + P × Page fault time

$$\text{EAT} = (1 - P) \times 100 + P \times 25 \times 10^6$$

$$= 100 - 100P + 25{,}000{,}000P = \mathbf{100 + 24999900P} \quad \textbf{Ans.}$$

COMPARISON OF THESE TECHNIQUES, TECHNIQUES FOR SUPPORTING THE EXECUTION OF LARGE PROGRAMS – OVERLAY, DYNAMIC LINKING AND LOADING, VIRTUAL MEMORY-CONCEPT, IMPLEMENTATION BY DEMAND PAGING ETC.

Q.77. Show the comparison between the paging, segmentation and paged segmentation techniques.

Ans. The memory management techniques such as paging, segmentation and paged segmentation differs in many aspects. While comparing these techniques, use the following considerations –

(i) Hardware Support – It is sufficient to use the simple base register or a pair of base and limit registers for the single and multiple partition schemes, whereas paging and segmentation need mapping tables to define the address map.

(ii) Performance – Due to the complexity of the memory management algorithm, the time required to map a logical address to physical address increases. For simple systems, we are required to compare or add to the logical address operations that are fast. Paging and segmentation works fast, if the table is implemented in fast registers.

(iii) Fragmentation – Multiprogrammed system will perform more efficiently if it has a higher level of multiprogramming. Suppose a set of process is given, then the level of multiprogramming can only be increased by packing more processes into memory. To accomplish this task, reduce the waste memory or fragmentation. The fixed sized allocation unit systems, such as the single-partition scheme and paging, suffer from internal fragmentation. The variable sized allocation unit systems, such as multiple-partition scheme and segmentation, suffer from external fragmentation.

(iv) Relocation – The solution of the external-fragmentation problem is compaction. It involves to shifting a program in memory without the program noticing the change. This is considered that at the time of execution the logical addresses be relocated dynamically. If address relocation is done at load time then compact storage cannot be done.

(v) Swapping – The operating system determines at several intervals that the CPU scheduling policies, processes are copied from main memory to a backing store and are copied back to main memory. It is the swapping process which allows more processes to be run than to fit into memory at one time.

(vi) Sharing – Sharing is the process through which the level of multiprogramming can be increased by sharing the code and data among different users. Sharing requires to use either paging or segmentation which provides small packets of information (pages or segments) that can be shared. Sharing means running many processes with a limited amount of memory, but shared programs and data must design carefully.

(vii) Protection – In paging or segmentation, the user program can be declared as execute only, read only, or read-write. It is the necessary restriction with shared code or data and is generally useful to provide simple run-time checks for common programming errors.

Q.78. Explain the overlay technique in the execution of large program.

Ans. In a general computing sense, overlaying means "the process of transferring a block of program code or other data into main memory, replacing what is already stored" overlaying is a programming method that allows programs to be larger than the computer's main memory. An embedded system would normally use overlays because of the limitation of physical memory, which is internal memory for a system-on-chip and the lack of virtual memory facilities.

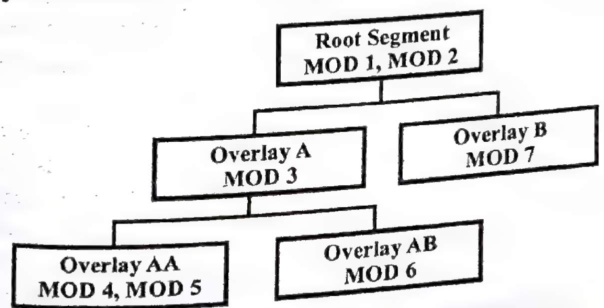


Fig. 3.82

Constructing an overlay program involves manually dividing a program into self-contained object code blocks called overlays laid out in a tree structure. Sibling segments,

those at the same depth level, share the same memory, called overlay region or destination region. An overlay manager, either part of the operating system or part of the overlay program, loads the required overlay from external memory into its destination region when it is needed. Often linkers provide support for overlays.

These statements define a tree consisting of the permanently resident segment, called the root, and two overlays A and B which will be loaded following the end of MOD2. Overlay A itself consists of two overlay segments, AA, and AB. At execution time overlays A and B will both utilize the same memory locations; AA and AB will both utilize the same locations following the end of MOD3.

All the segments between the root and a given overlay segment are called a path.

Applications – Most business applications are intended to run on platforms with virtual memory. A developer on such a platform can design a program as if the memory constraint does not exist unless the program's working set exceeds the available physical memory. Most importantly, the architect can focus on the problem being solved without the added design difficulty of forcing the processing into steps constrained by the overlay size. Thus, the designer can use higher-level programming languages that do not allow the programmer much control over size (e.g. Java, C++, Smalltalk).

Still, overlays remain useful in embedded systems. Some low-cost processors used in embedded systems do not provide a memory management unit (MMU). In addition many embedded systems are real-time systems and overlays provide more determinate response-time than paging. For example, the Space shuttle primary avionics system software (PASS) uses programmed overlays.

Even on platforms with virtual memory, software components such as codecs may be decoupled to the point where they can be loaded in and out as needed.

Q.79. What does dynamic linking and loading do ?

Ans. **Dynamic Linking** – The dynamic linking concept is as same as that of dynamic loading. The loading is being postponed until execution time, than the linking is postponed. This feature is used in system libraries such as language subroutine libraries. All the programs on a system have to make a copy of their language library if their is no linking facility. Due to this requirement the disk space and main memory both was wasted. With the help of dynamic linking a stub is included in the image for each library-routine reference. Here stub is a small piece of code which indicate how to locate the appropriate memory resident library routine, or the process of loading the library if the routine is not already present.

On the execution of this stub it checks the needed routine is already in memory or not. If not then the program loads the routine into memory. The address of the routine was replaced by the stub itself and executes the routine. Thus the code segment will reach in the next time and the library routine is executed directly, incurring no cost for dynamic linking. Hence all the processes under this scheme which use a language library will execute only one copy of the library code. The fig. 3.22 shows the dynamically linked libraries.

Dynamic Loading – As we know that the entire program and data of a process is loaded in physical memory for the process to execute. The process size is limited as compared to the size of physical memory. The ***dynamic loading*** concept is used to obtain the better memory-space utilization. In case of dynamic loading, the routine is not loaded until it is called. All the routines are kept on disk in the form of relocatable load format. The main program is loaded into memory and then it is executed. Whenever a routine needs to call another routine, the calling routine first check whether the other routine has been loaded. If it was not loaded then the relocatable linking loader called to load the desired routine into memory to update the program's address tables to reflect this change. Then the control is passed to the newly loaded routine.

The main advantage of dynamic loading is, it does not allow to load an unused routine. When large amount of lode are required to handle infrequently occurring cases such as error routines, the method of dynamic loading is used. In such cases, although the program size may be large but the portion that is used may be small in size.

Q.80. Explain the concept of virtual memory with example. ***(R.G.P.V., June 2004)***

Or

Explain virtual memory. ***(R.G.P.V., June 2017)***

Ans. Virtual memory is a technique that allows the execution of process that may not be completely in memory. In this only a portion of the virtual address space of resident process may actually be loaded into physical memory. As a consequence, the sum of the virtual address spaces of active processes in a virtual-memory system can exceed the capacity of the physical memory provided that the physical memory is large enough to hold a minimum amount of address space of each active process. Thus whereas the real-memory schemes strive to approach 100% utilization of physical memory, virtual-memory system routinely provide apparent utilization factors in excess of 100%. Also, the allowable size of the virtual-address space of a single process can exceed the maximum capacity of the physical memory installed in a given system.

This feat is accomplished by maintaining an image of the entire virtual address space of a process on secondary storage, and by bringing its sections into main memory when needed. The choice of the which sections to bring in, where to place them, and when to bring them is made by operating system. These decisions are influenced by the demand of the active processes and by the overall system resource availability.

The details of virtual-memory management are generally transparent to programmers. Programmers are practically relieved of the burden of trying to fit a program into limited memory. Moreover, the same program may run without reprogramming or recompilation on systems with significantly different capacities of installed memory In principle, in systems with virtual memory, only the execution speed and not the function of a given application is affected by the amount of installed physical memory.

In addition to separating logical memory from physical memory, virtual memory also allows files and memory to be shared by several different processes through page sharing.

For example, a 1 M program can run on a 256 K machine by choosing which 256 K to keep in memory at each instant, with pieces of the program being swapped between disk and memory as needed.

Virtual memory can also work in a multiprogramming system. For example, eight 1 M programs can each be allocated a 256 K partition in a 2 M memory, with each program operating as though it had its own, private 256 K machine.

Q.81. What do you mean by demand paging ? Explain its concept.

Or

Explain demand paging with example. ***(R.G.P.V., Dec. 2011)***

Or

Define demand paging. ***(R.G.P.V., June 2016)***

Or

Write short note on demand paging. ***(R.G.P.V., May 2018)***

Ans. A demand paging system is similar to a paging system with swapping as shown in fig. 3.83. When a process is to be executed, it is swapped in into memory. Rather than swapping the entire process into memory, a lazy swapper is used, which never swaps a page into memory unless that page will be needed.

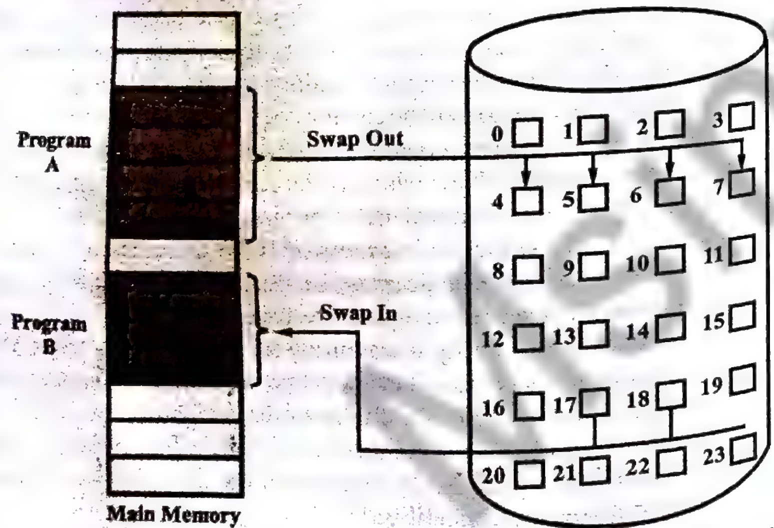


Fig. 3.83 Transfer of a Paged Memory of Contiguous Disk Space

Basic Concept – When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

This scheme needs some form of hardware support to distinguish between those pages that are in memory and those are on the disk.

The valid-invalid bit scheme can be used for this purpose. When this bit is set to valid, this value indicates that the associated page is both legal and in memory. If the bit is set to invalid, this value indicates that the page either is not valid, or is valid but is currently on the disk. The page table entry for a page that is brought into memory is set as usual, but the page table entry for a page that is not currently in memory is marked invalid, or contains the address of the page on disk as shown in fig. 3.84.

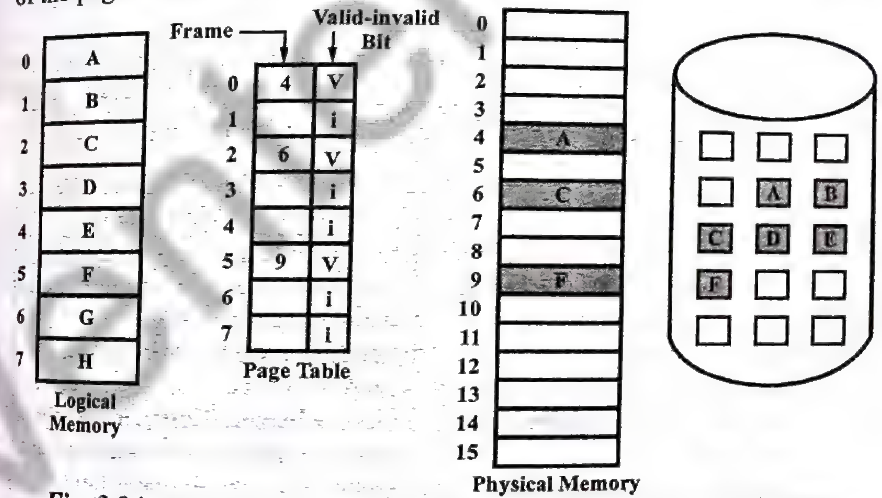


Fig. 3.84 Page Table when Some Pages are not in Main Memory

Q.82. Contrast demand paging versus working set model as page fetch policy. ***(R.G.P.V., Dec. 2013)***

Ans. Fetch policy deals with when to bring a page in the main memory. There are basically two methods in this regard, namely, demand paging and working set. They are described as follows –

(i) ***Demand Paging*** – Refer Q.81.

(ii) ***Working Set Model*** – Refer Q.71.

Q.83. What are the major problems to implement demand paging ? ***(R.G.P.V., Dec. 2012)***

Ans. Two major problems must be solved to implement demand paging –

(i) ***The Page Replacement Algorithm*** – When a page replacement is required, we must select which frames are to be used.

(ii) ***The Frame Allocation Algorithm*** – We must decide how many frames to allocate to each process.

Q.84. Write short note on the demand segmentation.

Ans. This scheme is just like demand paging but it is less efficient. This scheme is used where sufficient hardware is not available for demand paging.

In this scheme, memory is allocated in segments. A process is divided into segments. Only those segments of process are in memory which required to perform current execution. There is a table maintained that keeps the information of each segment like size of segment, location in memory and status that shows whether segment is in memory or not.

If CPU demands a segment, which is not in memory, then a segment fault occurs. Thus, required segment is brought in the memory. If sufficient memory is not available to keep this new segment, then segment replacement algorithm is used to swap-out a segment from memory and swap-in the newly arrived segment. Generally LRU algorithm is used.

Q.85. Discuss the difference between demand paging and demand segmentation. ***(R.G.P.V., June 2017)***

Ans. Difference between demand paging and demand segmentation are as follows –

S.No.	*Demand Paging*	*Demand Segmentation*
(i)	Demand paging is a concept by which no page is fatched until a request made for that page.	In demand segmentation, segments are fetched in the memory only when they are needed.
(ii)	Pages are of same size.	Segments are of variable size.
(iii)	More efficient.	Less efficient.
(iv)	Requires sufficient hardware to implement.	Used where sufficient hardware is not available for demand paging.
(v)	Generally, more page fault.	Generally, less segment fault.

NUMERICAL PROBLEMS

Prob.31. On a system using demand paged memory, it takes 200 ns to satisfy a memory request if the page is in memory. If the page is not in memory, the request takes 7 ms if a free frame is availabe or the page to be swapped out has not been modified. It takes 15 ms if the page to be swapped out has been modified. What is the effective access time (E.A.T.) if the page fault rate is 5% and 60% of the time the page to be replaced has been modified ? Assume the system is only running a single process and the CPU is idle during page swaps. ***(R.G.P.V., June 2011)***

Sol. The percentage of accesses satisfied in 200 ns is 95%. Of the 5% of accesses that result in a page fault, 40% require 7 ms. Thus, 5% × 40% = 2% of all accesses take 7 ms. Similarly, 5% × 60% = 3% of accesses take 15 ms. Converting all times to μs yields the following –

$$\text{Effective access time} = 0.95 \times 0.2 + 0.02 \times 7000 + 0.03 \times 15000$$
$$= 590.19\ \mu s$$

⌘⌘

UNIT 4

INPUT/OUTPUT – PRINCIPLES AND PROGRAMMING, INPUT/OUTPUT PROBLEMS, ASYNCHRONOUS OPERATIONS, SPEED GAP FORMAT CONVERSION

Q.1. Explain the various principles of programming and basic input-output functions.

Ans. The principles of programming are as follows –

(i) A program in C begins with # provides an instruction to the C preprocessor. It is executed before the actual compilation is done. #include and #define are the two most common directives. Identifies the header file for standard input and output needed by the printf( ).

(ii) The main( ) function identifies the start of the program and it cannot be used for any other variable.

(iii) The curly braces { } identify the body of a program and the start and end of a function.

(iv) An instruction that copies data from an input device into memory is called input operation. An instruction that displays information stored in memory to the output devices is called output operation.

(v) Some of the functions which perform input-output operations are printf( ), scanf( ), getchar( ) and putchar( ).

(vi) A series of continuous characters are treated as tokens by the C compilers. The tokens can be classified as reserved words, identifiers, constants, string literal, punctuators and operators.

(vii) In programming language reserved words are known as keyword that identify language entities such as statements, data types, language attributes etc.

(viii) Program identifiers are the words that are used to represent the program entities such variable name, function name etc. For example – caltot ( ), my_name etc.

(ix) There are four basic data types in programming – int, float, double and char.

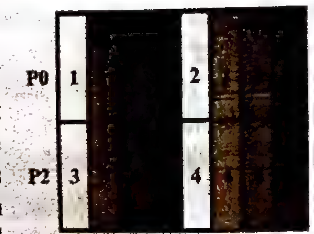
(x) In programming variable is the name associated with a memory cell whose value can change. The declaration of the variable shows the type of a variable such as int num = 5.

Q.2. Explain the different issues in compiling input/output intensive problems.

Ans. The input/output problems or the issues which occur in compiling the program can be defined as follows –

(i) File Space – The file space of the application depends on how the distributed data is stored in files. A set of variables is said to have *spatial locality in file space iff* if it lies in the same file. The definition of the locality in file space can be extended to take into account the data storage order. We define sequential file locality as follows – two elements are said to have sequential, file locality if they are stored at consecutive positions in the same file.

Consider fig. 4.1 (b). It shows the file corresponding to the global array which is distributed over four processors [fig. 4.1 (a)]. The data is stored in the file in the column-major order. The first sub-column of processor 1 is stored at the beginning of the file, followed by the first sub-column of processor 3 and so on. Therefore, the data belonging to a particular processor (or having spatial processor locality) may not be stored consecutively in the file (or may not exhibit sequential file locality). But elements of each sub-column will exhibit both spatial proce- ssor locality and sequential file locality.



(a) Processor Space

(b) File Space

Fig. 4.1

(ii) Disk Space – The disk space of the application depends on how the files are striped across disks. A set of variables is said to have *spatial locality in disk space iff* if it lies in the same disk. The definition of the locality in disk space can also be extended to take into account the data storage order. We can define sequential disk locality as follows – two elements are said to have sequential disk locality if they are stored at consecutive positions in the same disk.

Consider fig. 4.2. It shows how the global files are striped across four disks in a round-robin fashion. Each disk contains sub-columns of different processors. For example, disk D0 contains first and third sub-columns of processors 1 and 2, disk D1 contains first and third sub-columns of processors 3 and 4 and so on. The elements of the first and third sub-columns of processor 1 possess both spatial processor and disk locality since they belong to same processor as well as same disk. These elements also exhibit spatial file locality because they belong to the same file. However, these elements do not have sequential locality in file space since these sub-columns are stored at non-consecutive positions in the global file. Similarly, only the elements of the first sub-column will exhibit sequential disk locality, since they will be stored at consecutive positions on the disk D0. This example shows that data may exhibit file locality but not disk locality and vise versa.

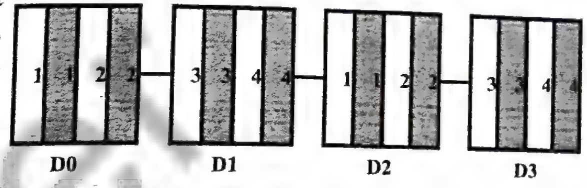


Fig. 4.2 Disk Space

Q.3. Write a short note on asynchronous operations.

Ans. Asynchronous operation or asynchronous working is where a sequence of operations is executed such that the operations are executed out of time coincidence with any event. It can also be an operation that occurs without a regular or predictable time relationship to a specified event, e.g. the calling of an error diagnostic routine that may receive control at any time during the execution of a computer program.

As we know that the send, receive and reply operations can be synchronous or asynchronous. The synchronous operation blocks a process till the operation is completed whereas the asynchronous operation is known as non-blocking and only initiates the operation. Remember that the synchronous and asynchronous are known as blocking and not blocking operations but its not vice versa. It means not every blocking operation is synchronous but not every non blocking operation is asynchronous. For example, in case of asynchronous it sends the blocks till the receiver machine has received the message is blocking but in case of synchronous it is not true since the receiver process may not have received it. These definitions of synchronous and asynchronous operations are similar but not identical, which tends to equate synchronous with blocking.

In message passing mechanism, asynchronous allows more parallelism. Until the process does not block the asynchronous do some computation while the message is in transit. The process can express their interest in receiving messages on multiple ports simultaneously, whereas in the case of synchronous system the parallelism can be occurred by forking a separate process for each concurrent operations, but this approach results the cost of extra process management.

Q.4. What do you understand by speed gap format conversion in an operating system ?

Ans.

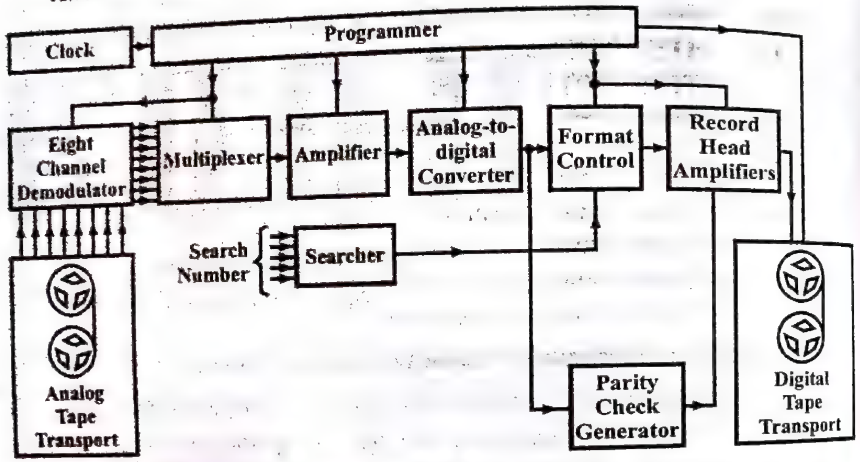


Fig. 4.3 Block Diagram of Central Converting and Recording System

The above diagram shows the central data converting system. It is a versatile data handling system which is used to play back prerecorded analog magnetic tapes and convert the signals on the analog tapes into a digitally coded form. It helps to record the digital data on magnetic tape in the compatible format with an IBM computer. This system contains three main functions – analog-to-digital conversion, event number conversion and automatic tape search.

While performing the analog-to-digital conversion, the central system accepts the magnetic tapes prepared by the remote system. When analog signal played back are demodulated by an eight channel demodulator. Then the high-speed, solid-state, diode bridge multiplexer scans the amplifier outputs in a predetermined manner. The voltage which was produced at the output of the multiplexer is sequentially converted to a 3-digit binary coded decimal number with the help of analog-to-digital converter.

The 3-digit equivalents of the analog inputs are fed to the format of control circuits. These types of data format of the system is required by the IBM computers. In such computers, the digital data is separated into records. This record is known as the block of data, followed by a blank space, which is called a record gap.

The analog data are continuously digitized, due to these record gaps the resulting digital data recorded discontinuously. It is accomplished by selecting the recording speed and conversion rate so that the end-of-record gap is placed after every 2 samples of each 8 inputs. The storage is used for the samples that are taken during the record gap.

The digital magnetic tape recorder, records various speeds, which results in various sampling rates for all eight channels. By changing the ratio of recording to playback speeds of the analog recorder increases the sampling rates.

I/O INTERFACES, PROGRAMME CONTROLLED I/O, INTERRUPT DRIVEN I/O, CONCURRENT I/O

Q.5. Describe I/O interface.

Ans. Input output interface provides a method for transferring information between internal storage and external I/O devices.

Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.

The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.

The major differences are –

(i) Peripherals are electromechanical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.

(ii) The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.

(iii) Data codes and formats in the peripherals differ from the word format in the CPU and memory.

(iv) The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervises and synchronizes all input and out transfers.

These components are called interface units because they interface between the processor bus and the peripheral devices.

The I/O BUS and Interface Module defines – It the typical link between the processor and several peripherals.

The I/O bus consists of data lines, address lines and control lines.

The I/O bus from the processor is attached to all peripherals interface.

To communicate with a particular device, the processor places a device address on address lines.

Each interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller.

It is also synchronizes the data flow and supervises the transfer between peripheral and processor.

Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing.

The control lines are referred as I/O command. The commands are as following –

(i) Control Command – A control command is issued to activate the peripheral and to inform it what to do.

(ii) Status Command – A status command is used to test various status conditions in the interface and the peripheral.

(iii) Data Output Command – A data output command causes the interface to respond by transferring data from the bus into one of its registers.

(iv) Data Input Command – The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register.

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O –

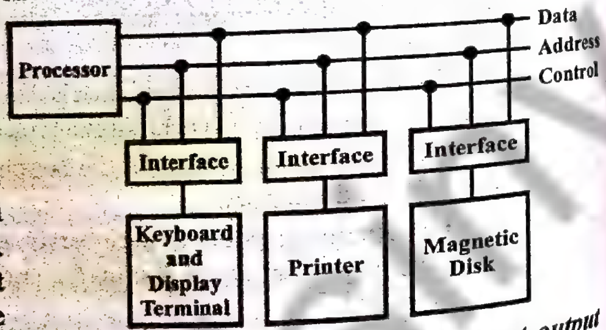


Fig. 4.4 Connection of I/O Bus to Input-output Devices

(i) Use two separate buses, one for memory and other for I/O.

(ii) Use one common bus for both memory and I/O but separate control lines for each.

(iii) Use one common bus for memory and I/O with common control lines.

Q.6. Explain the working process of programme input/output method.

Ans. Programmed Input/Output (PIO) is a method of transferring data between the CPU and a peripheral, such as a network adapter or an ATA storage device. Each data item transfer is initiated by an instruction in the program, involving the CPU for every transaction. In contrast, in direct memory access (DMA) operations, the CPU is not involved in the data transfer.

The term programmed I/O can refer to either memory-mapped I/O (MMIO) or Port-mapped I/O (PMIO). PMIO refers to transfers using a special address space outside of normal memory, usually accessed with dedicated instructions, such as IN and OUT in x86 architectures. MMIO refers to transfers to I/O devices that are mapped into the normal address space available to the program. PMIO was very useful for early microprocessors with small address spaces, since the valuable resource was not consumed by the I/O devices.

The best known example of a PC device that uses programmed I/O is the ATA interface; however, this interface can also be operated in any of several DMA modes. Many older devices in a PC also use PIO, including legacy serial ports, legacy parallel ports when not in ECP mode, the PS/2 keyboard and mouse ports, legacy MIDI and joystick ports, the interval timer, and older network interfaces.

The PIO interface is grouped into different modes that correspond to different transfer rates. The electrical signaling among the different modes is similar only the cycle time between transactions is reduced in order to achieve a higher transfer rate. All ATA devices support the slowest mode, mode 0. By accessing the information registers (using mode 0) on an ATA drive, the CPU is able to determine the maximum transfer rate for the device and configure the ATA controller for optimal performance.

The PIO modes require a great deal of CPU overhead to configure a data transaction and transfer the data. Because of this inefficiency, the DMA (direct memory access) (and eventually UDMA) interface was created to increase performance. The simple digital logic required to implement a PIO transfer still makes this transfer method useful today, especially if high transfer rates are not required like in embedded systems, or with FPGA chips where PIO mode can be used without significant performance loss.

Q.7. Give a brief overview of interrupt driven input/output.

Ans. Interrupt driven I/O is an alternative scheme dealing with I/O. Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupt. Relative to the total interrupt system, the processors enter an interrupt service routine. The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor. The interrupt technique requires more complex hardware and software, but makes far more efficient use of the computer's time and capacities. Fig. 4.5 shows the simple interrupt processing.

For input, the device interrrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports or memory mapping. For output,

the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Here the CPU works on its given tasks continuously. When an input is available, such as when someone types a key on the keyboard, then the CPU is interrupted from its work to take care of the input data. The CPU can work continuously on a task without checking the input devices, allowing the devices themselves to interrupt it as necessary.

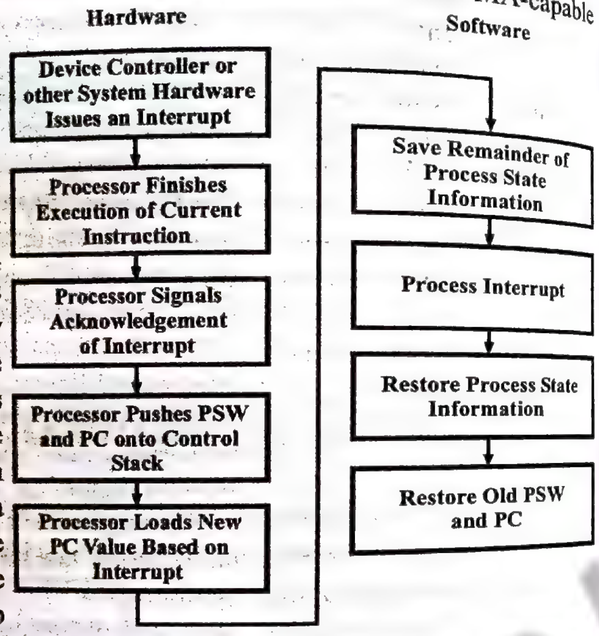


Fig. 4.5 Simple Interrupt Processing

Basic Operations of Interrupt –

(i) CPU issues read command.

(ii) I/O module gets data from peripheral whilst CPU does other work.

(iii) I/O module interrupts CPU.

(iv) CPU requests data.

(v) I/O module transfers data.

Advantages –

(i) Fast

(ii) Efficient.

Disadvantages –

(i) Can be tricky to write if using a low level language

(ii) Can be tough to get various pieces to work well together

(iii) Usually done by the hardware manufacturer/OS maker, e.g. Microsoft.

Q.8. What is the role of concurrent input/output in operating system ?

Ans. On AIX operating systems, you can use concurrent I/O in addition to direct I/O for chunks that use cooked files. Concurrent I/O can improve performance, because it allows multiple reads and writes to a file to occur concurrently, without the usual serialization of noncompeting read and write operations.

Concurrent I/O can be especially beneficial when you have data in a single chunk file striped across multiple disks.

Concurrent I/O, which you enable by setting the DIRECT_IO configuration parameter to 2, includes the benefit of avoiding file system buffering and is subject to the same limitations and use of KAIO as occurs if you use direct I/O without concurrent I/O. Thus, when concurrent I/O is enabled, you get both unbuffered I/O and concurrent I/O.

If informix uses concurrent I/O for a chunk, and another program (such as an external backup program) tries to open the same chunk file without using concurrent I/O, the open operation will fail.

Informix does not use direct or concurrent I/O for cooked files used for temporary dbspace chunks.

CONCURRENT PROCESSES – REAL AND VIRTUAL CONCURRENCY, MUTUAL EXCLUSION, SYNCHRONIZATION, INTER-PROCESS COMMUNICATION

Q.9. What is concurrent processes ?

Ans. Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means something that happens at the same time as something else. Tasks are broken down into subtasks that are then assigned to separate processors to perform simultaneously, instead of sequentially as they would have to be carried out by a single processor. Concurrent processing is sometimes said to be synonymous with parallel processing.

Q.10. Explain the term real and virtual concurrency in concurrent processing.

Ans. In concurrent system, if each task executes on a different processor then it is called the real concurrency. Alternatively, the concurrency is virtual if the tasks are interleaved. There are three types of concurrent systems –

(i) Multiprogramming Environment – In multiprogramming environment, there are multiple tasks shared by one processor. Whereas a virtual concurrency can be achieved by operating system, if the processor is allocated to each individual task, so that the virtual concurrency can appear if each task has a dedicated processor. The multiprogramming environment is shown in fig. 4.6.

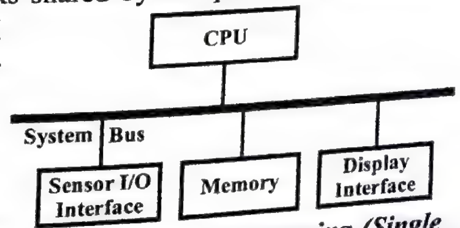


Fig. 4.6 Multiprogramming (Single CPU) Environment

(ii) Multiprocessing Environment – In multiprocessing environment two or more processors are used with shared memory. Only one virtual address space is used, which is common to all the processors. All the tasks are resided in shared memory. In this environment, real concurrency is supported as the processors executing concurrently. The tasks executing on different processors is done with each other via the shared memory. The multiprocessing environment is shown in fig. 4.7.

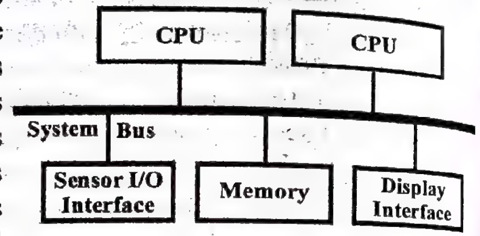


Fig. 4.7 Multiprocessing Environment

(iii) Distributed Processing Environment – Two or more computers are connected to each other by a communications network or high speed bus in distributed processing environment. There is no shared memory between the processors and each computer has its own local memory. Hence a distributed application consisting of concurrent tasks, which are distributed over the network communication via messages. The distributed processing environment is shown in fig. 4.8.

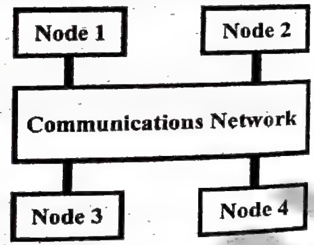


Fig. 4.8 Distributed Processing Environment

Q.11. Explain the term mutual exclusion with example.
(R.G.P.V., Dec. 2011)

Or

Briefly explain mutual exclusion. ***(R.G.P.V., Dec. 2016)***

Ans. Updating of a shared variable may be regarded as a critical section. The critical section is a sequence of instructions with a clearly marked beginning and end. It safeguards updating of one or more shared variables. When a process enters a critical section, it must complete all instructions therein before another process is allowed to enter the same critical section. Only the process executing the critical section is allowed to access the shared variable, all other processes should be prevented from doing so until the completion of the critical section. This is referred to as ***mutual exclusion.*** It permits a single process to temporarily exclude all others from using a shared resource in order to ensure the system's integrity.

If the shared resource is a variable, mutual exclusion ensures that at most one process at a time has access to it during the critical updates that lead to temporarily inconsistent values. Consequently, other processes see only consistent values of shared variables.

A solution to the mutual exclusion problem should –

(i) Ensure mutual exclusion between processes accessing the protected shared resource.

(ii) Make no assumptions about relative speeds and priorities of contending processes.

(iii) Guarantee that crashing or terminating of any process outside of its critical section does not affect the ability of other contending processes to access the shared resource.

(iv) When more than one process tries to enter the critical section, grant entrance to one of them in finite time.

Q.12. What is process synchronization ?

Ans. A cooperating process is one that can affect or be affected by other processes executing in the system. Cooperating processes may either directly share a logical address space, or be allowed to share data only through files. Concurrent access to shared data may result in data inconsistency. Process synchronization is a mechanism that ensures the orderly execution of cooperating processes that share a logical address space, so that data consistency is maintained.

Q.13. How process synchronization is achieved ?

Ans. Given a collection of cooperating sequential processes that share data, mutual exclusion must be provided. One solution is to ensure that a critical section of code is in use by only one process or thread at a time. Different algorithms exist for solving the critical-section problem, with the assumption that only storage interlock is available.

The main disadvantage of these user-coded solutions is that they all require busy waiting. Semaphores overcome this difficulty. Semaphores can be used to solve various synchronization problems and can be implemented efficiently, specially if hardware support for atomic operations is available.

Q.14. How is interprocess communication achieved in O.S. ?
(R.G.P.V., Nov./Dec. 2007)

Or

What is inter processes communication ? ***(R.G.P.V., June 2016)***

Ans. A way for the operating system to provide the means for cooperating processes to communicate with each other via an interprocess communication (IPC) facility.

IPC provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space. IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network. An example is a chat program used on the world wide web.

IPC is best provided by a message-passing system. The function of a message system is to allow processes to communicate with one another without the need

to resort to shared data. We know that the message-passing is used as a method of communication in ***microkernels.*** In this scheme, services are provided as ordinary user processes. That is the services operate outside of the kernel. Communication among the user processes is accomplished through the passing of messages. An IPC provides at least the two operations – send (message) and receive (message).

Messages sent by a process can be of either fixed or variable size. If only ***fixed-sized messages*** can be sent, the system-level implementation is straightforward. This restriction, however, makes the task of programming more difficult, on the other hand, ***variable-sized messages*** require a more complex system-level implementation, but the programming task becomes simpler.

If processes P and Q want to communicate, they must send messages to and receive message from each other, a communication link must exist between them. This link can be implemented in a variety of ways. Here are several methods for logically implementing a link and the send/receive operations –

(i) Direct or indirect communication
(ii) Symmetric or asymmetric communication
(iii) Automatic or explicit buffering
(iv) Fixed-sized or variable-sized messages
(v) Send by copy or send by reference.

Q.15. Discuss various design issues related to message-passing systems.

Ans. The actual function of a message-passing is provided in the form of a pair of primitives –

send (destination, message)
receive (source, message)

A process sends information in the form of a message to another process designated by a destination. A process receives information by executing the receive primitive.

A number of design issues relating to message-passing systems are –

(i) Naming – The naming can be of two types – direct and indirect.

(a) Direct Naming – Direct naming means that when invoking a message operation, each sender must name the specific recepient, and conversely, each receiver must name the source from which it wishes to receive a message. For example, if process A wants to send a message to process B, the following statements may be necessary –

```
process A;
. . .
    send (B, message);
    . . . .
process B;
. . .
    receive (A, message);
```


where message is the actual message transmitted, and A and B are identities of the sender and receiver of the message, respectively.

This type of message communication is symmetric in the sense that each sender must know and name its receivers, and vice versa.

(b) Indirect Naming – Indirect naming means that messages are sent to and received from specialized repositories dedicated to that purpose. The repositories are called mailboxes due to their way of functioning. For example, if process A wants to send a message to process B through mailbox 1, the following statements may be necessary –

```
process A;
. . . .
    send (mailbox1, message);
    . . . .
process B;
    . . . .
    receive (mailbox1, message);
```

The first operation places the produced message into the named mailbox, mailbox1, and the second operation removes a message from the mailbox 1 and provides it to the receiving process through private variable message. This form of message communication requires some system services for maintenance of mailboxes such as create_mailbox and delete_mailbox.

(ii) Copying – Message exchange between two processes, transfers the contents of the message from the sender's to the receiver's addressing space. This may be accomplished by either copying the whole message into the receiver's addressing space, or by passing a pointer to the message between two processes. It means that message passing can be by value or by reference.

When a message facility is implemented in such a way that the operating system actually copies the message from sender's to receiver's space, the two processes remain fully decoupled from each other. Essentially, an imprint of the sender's data is made available to the receiver. It is not the matter that what the receiver does with its copy, the original remains unaffected on the sender's space.

On the other side, the copying process has a negative effect because copying messages consume memory and processor cycles. The memory protection and asynchronous message communication schemes require each message should be first copied to the buffer of the operating system from the sender's space, and then copied to the receiver's space from the buffer. Then it is clear that

Fig. 4.9 Copying of the Message

double copying is occured to deliver a single copy, first in the buffer and second in the receiver's space, instead of directly into receiver space. It is actually time consuming. The double copying is illustrated in the fig. 4.9.

(iii) Synchronous Versus Asynchronous Message Exchange – Generally the exchange of message between sender and receiver can be either synchronous or asynchronous. The detail discussion of both methods are as follows –

(a) Synchronous Message Exchange – When a message exchanged is synchronous, both the sender and the receiver must come together to complete the transfer. In this method, the SEND operation is blocked, means, when a sender wants to send a message for which RECEIVE is not issued, the sender must be suspended until the willing receiver accepts the message. In other words, if any sender process wants to send any message, then the operation is accomplished only if the receiver process is ready to accept the message.

(b) Asynchronous Message Exchange – In this method, the sender is not blocked if there RECEIVE is not issued, like synchronous message exchange method. The asynchronous, nonblocking SEND is already implemented by having operating system buffer to store the message, until the matching RECEIVE is issued. Eventually the sender process may continue execution after sending a message and no need to be suspended, regardless of the activity of receivers.

(iv) Message Length – Messages can be of fixed or variable size. The trade-off is one of overhead versus flexibility.

Fixed size messages result in lower overhead by virtue of allowing the related system buffers to be of fixed size as well, which makes their allocation quite simple and efficient. The problem is that messages come in various sizes, and fitting them into smaller chunks of fixed size results in additional overhead. For example, short messages waste buffer space, whereas long messages must be split and sent in installments, which may cause sequencing problems at the receiving end. The variable-size messages alleviates these problems by dynamically creating buffers to fit the size of each individual message. But the problem with this is the management of the operating system's dynamic memory pool. Allocation of variable-size chunks is costly in terms of CPU time and may lead to memory fragmentation.

(v) Queuing Discipline – The simplest queuing discipline is first-in-first-out, but this may not be sufficient if some messages are more urgent than others. An alternative is to allow the specifying of message priority, on the basis of message type or by designation by the sender. Another alternative is to allow the receiver to inspect the message queue and select which to receive next.

Q.16. What is race condition ? How is it caused ? What are implications of having a race condition ? ***(R.G.P.V., June 2010)***

Or

Explain the term race condition with example. ***(R.G.P.V., Dec. 2011)***

Ans. In some operating system, some processes share data for reading and writing then the result of operation depends on the order in which writing and reading take place. This situation is called ***race condition***. The share data may be in main memory or it may be a shared file, the location of the shared memory does not change the nature of the communication or the problems that arise.

Now consider a simple example, a print spooler. When a process wants to print a file, it enters the file name in a special spooler directory. Another process, the printer daemon, periodically checks to see it there are many files to be printed, and if there are it prints them and then removes their names from the directory. Imagine that our spooler directory has a large number of slots, numbered, 0, 1, 2, ... each one capable of holding a file name. Also imagine that there are two shared variables, ***out*** which points to the next file to be printed, and ***in***, which points to the next free slot in the directory. These two variables might well be kept on a two-word file available to all processes. At a certain instant, slots 0 to 3 are empty and slots 4 to 6 are full. More or less concurrently processes A and B decide they want to queue a file for printing (fig. 4.10).

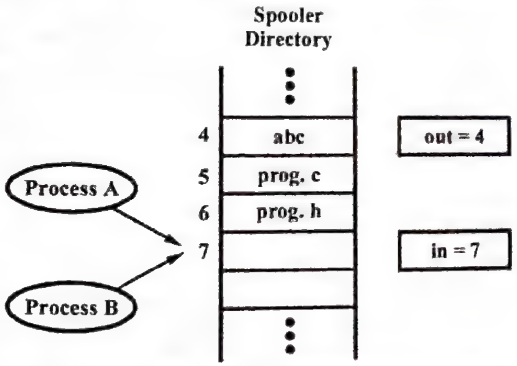


Fig. 4.10 Two Process Wants to Access Share Memory at the Same Time

In fig. 4.10, process A read ***in*** and stores 7, in a local variable called ***next_free_slot***. Just then a clock interrupt occurs and the CPU decides that process A has run long enough, so it switches to process B. Process B also reads ***in***, and also gets a 7, so it stores the name of its file in slot 7 and updates ***in*** to be an 8. Then it goes off and does other things.

Eventually process A runs again, starting from place it left off. It looks at ***next_free_slot*** finds a 7 there, and writes its file name in slot 7, erasing the name that process B just put there. Then it computes ***next_free_slot*** + 1, which is 8, and sets ***in*** to 8. The spooler directory is now internally consistent, so the printer daemon will not notice anything wrong, but process B will never get any output. Situations like this are called ***race*** conditions. Debugging programs containing race condition is no fun at all. The results of most test runs are fine, but once in a rare while something weird and unexplained happens.

CRITICAL SECTION PROBLEM, SOLUTION TO CRITICAL SECTION PROBLEM – SEMAPHORES – BINARY AND COUNTING SEMAPHORES, WAIT AND SIGNAL OPERATIONS AND THEIR IMPLEMENTATION

Q.17. Explain critical section problem. ***(R.G.P.V., June 2011)***

Or

Explain the term critical section with example. ***(R.G.P.V., Dec. 2011)***

Or

Define the term critical section. ***(R.G.P.V., Dec. 2014)***

Or

Briefly explain critical section problem. ***(R.G.P.V., Dec. 2016)***

Or

State and explain critical section problem. ***(R.G.P.V., June 2017)***

Ans. A system consisting of n processes $\{P_0, P_1, ..., P_{n-1}\}$. Each process has a segment of code, called a ***critical section***, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of this system is that when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is ***mutually exclusive*** in time. The critical section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is ***entry section***. The critical section may be followed by an ***exit section***. The remaining code is the ***remainder section***.

```
do {
        entry section
            critical section
        exit section
            remainder section
   } while (1);
```

Fig. 4.11 General Structure of a Typical Process P_i

Q.18. What are the requirements of a solution to critical section problem? Write solution to critical section problem.

Ans. A solution to the critical section problem must satisfy the following three requirements –

(i) Mutual Exclusion – If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

(ii) Progress – If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.

(iii) Bounded Waiting – There exist a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Two Process Solutions – The processes are numbered P_0 and P_1. For convenience, when presenting P_i, we use P_j to denote the other process, that is $j = = 1 - i$.

Algorithm 1 – Let the processes share a common integer variable turn initialized to 0 or 1. If turn = = i, then process P_i is allowed to execute in its critical section. The structure of process is shown in fig. 4.12.

```
do {
        while (turn ! = i);
            critical section
        turn = j;
            remainder section
   } while (1);
```

Fig. 4.12 The Structure of Process P_i in Algorithm 1

This solution ensures that only one process at a time can be in its critical section. However, it does not satisfy the progress requirement, since it requires strict alteration of processes in the execution of the critical section. For example, if turn = = 0 and P_1 is ready to enter its critical section, P_1 cannot do so, even though P_0 may be in its remainder section.

Algorithm 2 – The problem with algorithm 1 is that it does not retain sufficient information about the state of each process; it remembers only which process is allowed to enter its critical section. To remedy this problem, we replace the variable turn with the following array

boolean flag [2];

The elements of the array are initialized to false – If flag [i] is true, this indicates that P_i is ready to enter the critical section. The structure of process P_i is shown in fig. 4.13.

```
do
   {
      flag [i] = true ;
      while (flag [j]) ;
      critical section
      flag [i] = false ;
      remainder section
   }  while (1) ;
```

Fig. 4.13 The Structure of Process P_i in Algorithm 2

In this algorithm, process P_i first sets flag [i] to be true, i.e., it is ready to enter its critical section. Then, P_i checks to verify that process P_j is not also

ready to enter its critical section. If P_j is ready, then P_i will wait until P_j has indicated that it no longer needs to be in the critical section. At this point P_i will enter the critical section. On exiting the critical section, P_i will set flag [i] to be false, allowing the other process to enter its critical section.

This solution meets the mutual-exclusion requirement. However, the progress requirement is not met. For instance, consider the following execution sequence –

T_0 : P_0 sets flag [0] = true

T_1 : P_1 sets flag [1] = true

Now P_0 and P_1 are looping forever in their respective while statements.

Algorithm 3 – A correct solution to the critical section problem is obtained by using the ideas of both algorithm 1 and algorithm 2, where all three requirements are met. The processes share two variables –

boolean flag [2] ;

int turn ;

Initially flag [0] = flag [1] = false, and the value of turn is either 0 or 1. The structure of process P_i is shown in fig. 4.14.

```
do {
        flag [i] = true ;
        turn = j ;
        while (flag [i] && turn = = j);
            critical section
        flag [i] = false ;
            remainder section
   } while [1] ;
```

Fig. 4.14 The Structure of Process P_i in Algorithm 3

In this algorithm, process P_i first sets flag [i] to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section it can do so. If both processes try to enter at the same time, turn will be set to both i and j at the same time. Only one of these assignments last; the other will occur, but will be overwritten immediately. The eventual value of turn decides which of the two processes is allowed to enter its critical section first.

Q.19. What is a semaphore ? ***(R.G.P.V., Dec. 2012)***

Or

Explain about semaphore. ***(R.G.P.V., Dec. 2014)***

Ans. The Dekker's algorithm and Peterson's algorithm to solve the mutual exclusion are not easy to generalize to more complex problems. A synchronization tool called a ***semaphore***, is used to overcome this difficulty. A semaphore S is an integer variable that, apart from initialization, is accessed by two standard atomic operations wait and signal. These operations were originally termed P (for wait to test) and V (for signal to increment) by Dijkstra. These two operations take one argument each – the semaphore variable.

The classical definition of wait in pseudocode is

```
wait (S) {
    while (S ≤ 0)
        ; // no-op
    S – –;
}
```

The classical definition of signal in pseudocode is

```
signal (S) {
    S + +;
}
```

Modifications to the integer value of the semaphore in wait and signal operations must be executed indivisibly. It means that when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value. In addition, in the case of the wait (S), the testing of the integer value of S ($S \leq 0$), and its possible modification (S – –) must be executed without interruption.

Q.20. Write the usage of semaphore.

Ans. Semaphores can be used to deal with the n-process critical section problem. The n processes share a semaphore ***mutex*** (standing for mutual exclusion), initialized to 1. Each process P_i is organized as shown in fig. 4.15.

```
do  {
      wait (mutex);
      critical section
      signal (mutex);
      remainder section
    } while (1);
```

Fig. 4.15 Mutual Exclusion Implementation with Semaphores

Semaphores can be used to solve various synchronization problems. For example, there are two concurrently running process – P_1 with a statement S_1 and P_2 with a statement S_2. If it is required that S_2 be executed only after S_1 has completed. This scheme can be implemented by letting P_1 and P_2 share a common semaphore synch, initialized to 0, and by inserting the statements.

S_1;

signal (synch);

in process P_1, and the statements

wait (synch);

S_2;

in process P_2. Because synch is initialized to 0, P_2 will execute S_2 only after P_1 has invoked signal (synch), which is after S_1.

Q.21. What is critical section problem and explain two process solutions and multiple process solutions ? ***(R.G.P.V., Dec. 2012)***

Ans. **Critical Section Problem** – Refer Q.17.

Two Process Solution – Refer Q.18.

Multiple Process Solution – Refer Q.20.

Q.22. Define binary semaphores.

Ans. A binary semaphore is a semaphore with an integer value 0 or 1. A binary semaphore is easier to implement than a counting semaphore (also called general semaphore).

If S is a counting semaphore, it can be implemented in terms of binary semaphores using the following data structures.

```
binary-semaphore S1, S2;
int C;
```

Initially S1 = 1, S2 = 0, and the value of integer C is set to the initial value of the counting semaphore S.

The wait operation on the counting semaphore S can be implemented as follows –

```
wait (S1);
C – –
if (C < 0)
{
    signal (S1);
    wait (S2);
}
signal (S1);
```

The signal operation on the counting semaphore S can be implemented as follows –

```
wait (S1);
C + +;
if (C < = 0)
    signal (S2);
else
    signal (S1);
```

Q.23. What is hard and soft semaphore ? (R.G.P.V., Dec. 2015, June 2016)

Ans. For semaphores, a queue is used to hold processes waiting on the semaphore. The FIFO is the fairest policy in which processes are removed from such a queue. It means the process that has been blocked the longest is released from the queue first. A semaphore whose definition includes this policy is called a hard semaphore. A semaphore that does not specify the order in which processes are removed from the queue is called a soft semaphore.

Q.24. Give a solution that defines the mutual exclusion for two processes using semaphore.

Ans. The following program is written in Pascal is using a semaphore variable *s* to achieve mutual exclusion for two processes.

```
Program mutual_exclusion_sema
        var s : semaphore ;
        procedure processone;
        while true do
            begin
                P(s);
                criticalsectionone;
                V(s);
                otherstuffone ;
            end
        end;
        procedure processtwo;
        begin
            while true do
                begin
                    P(s);
                    criticalsectiontwo ;
                    V(s);
                    otherstufftwo;
                    end
        end;
        begin
            s =1 ;
            parbegin
            processone;
            processtwo;
            parend
        end;
```

The semaphore variable *s* is initialized to 1. Both processes are started simultaneously. We are assuming that speed of ***processone*** is greater than ***processtwo***. The operation P(s) i.e., P (1) is executed in first procedure and it decrements the value of *s* and set *s* = *0* .Then ***processone*** goes in its critical region.

After ***processone*** exits from its critical region, it executes V(s) i.e., V(0) which increments *s* by 1 that allows ***processtwo*** to execute.

Now when ***processone*** is executing in its critical region, if ***processtwo*** wants to enter in its critical region then it cannot enter, since that statement P(s) in first procedure have changed the value of *s* to 0 and when statement P(s) in second procedure is executed, it puts the ***processtwo*** into waiting state, since *s* = *0*. So mutual exclusion has been achieved. In the case when both processes have same speed, they will execute the P(s) operation at same

time but s is a semaphore variable so only one can access it, so mutual exclusion preserved here.

Q.25. Explain WAIT and SIGNAL operations with their implementation.

Or

Explain why 'wait' and 'signal' operations on a semaphore should be executed automatically. *(R.G.P.V., May 2018)*

Ans. Two standard operations, wait and signal are defined on the semaphore. Entry to the critical section is controlled by the wait operation and exit from a critical region is taken care by signal operation. The wait, signal operations are also called P and V operations. The manipulation of semaphore (S) takes place as following –

(i) The wait command P(S) decrements the semaphore value by 1. If the resulting value becomes negative then P command is delayed until the condition is satisfied.

(ii) The V(S), i.e. signals operation increments the semaphore value by 1.

Mutual exclusion on the semaphore is enforced within P(S) and V(S). If a number of processes attempt P(S) simultaneously, only one process will be allowed to proceed and the other processes will be waiting. These operations are defined as under –

```
P(S) or wait (S) ;
If S > θ then
    Set S to S-1
Else
    Block the calling process (i.e. wait on S)

V(S) or signal(S);
If any processes are waiting on S
    Start one of these processes
Else
    Set S to S+1
```

The semaphore operation are implemented as operating system services and so wait and signal are atomic in nature, i.e. once started, execution of these operations cannot be interrupted.

Thus semaphore is a simple yet powerful mechanism to ensure mutual exclusion among concurrent processes.

Q.26. Explain bounded buffer problem of synchronization. *(R.G.P.V., June 2010)*

Ans. The general statement is that there are one or more producers processes generating some type of data (records, characters) and placing these in a buffer. There is a single consumer that is taking items out of the buffer one at a time. The system is to be constrained to prevent the overlap of buffer operations. That is, only one agent (producer or consumer) may access the buffer at any one time. The bounded buffer producer-consumer problem assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full. The buffer is treated as a circular storage as shown in fig. 4.16, and pointer values must be expressed modulo the size of the buffer.

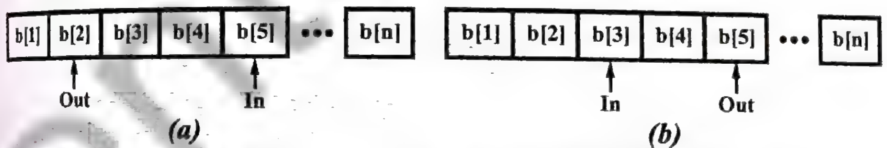


Fig. 4.16 Finite Circular Buffer for the Producer/Consumer Problem

Q.27. Write the solution of bounded buffer producer/consumer problem.

Ans. The producer and consumer functions can be expressed as follows (variable in and out are initialized to 0) –

```
producer –
while (true)
{
    /* produce item v */
    while ((in + 1)% n = = out)
        /* do nothing */;
        b[in] = v;
        in = (in + 1)% n;
}
consumer –
while (true)
{
    while (in = = out)
    /* do nothing */;
    w = b [out];
    out = (out + 1)% n;
    /* consume item w */;
}
```

Fig. 4.17 shows a solution using general semaphores (or counting semaphores). The semaphore e has been added to keep track of the number of empty spaces.

```
/* program boundedbuffer */
const int sizeofbuffer = /* buffer size*
semaphore s = 1;
semaphore n = 0;
semaphore e = sizeofbuffer;
void producer( )
{
    while (true)
    {
        produce( );
        wait(e);
        wait(s);
        append( );
        signal(s);
        signal(n)
    }
}
void consumer( )
{
    while (true)
    {
        wait(n);
        wait(s);
        take( );
        signal(s);
        signal(e);
        consume( );
    }
}
void main( )
{
    parbegin (producer, consumer);
}
```

Fig. 4.17 A Solution to the Bounded Buffer Producer/Consumer Problem using Semaphores

Hence, both producer and consumer processes cannot access the buffer at the same time so mutual exclusion is guaranteed.

Q.28. Write the implementation of semaphores without busy waiting. Write the solution for bounded buffer producer/consumer problem using semaphores.

(R.G.P.V., June 2007)

Ans. **Implementation of Semaphores without Busy Waiting** – The need for busy waiting can be overcome by modifying the definition of the *wait* and *signal* semaphore operations. When a process executes the wait operation and finds that the semaphore value is not positive, it must wait. However, rather than busy waiting, the process can block itself. The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.

A process that is blocked, waiting on a semaphore s, should be restarted when some other process executes a signal operation. The process is restarted by a ***wakeup*** operation, which changes the process from the waiting state to the ready state.The process is then placed in the ready queue.

To implement semaphores under this definition a semaphore can be defined as a C struct as follows –

```
typedef struct
{
    int value;
    struct process*L;
} semaphore;
```

Each semaphore has an integer value and a list of processes. When a process must wait on a semaphore, it is added to the list of processes. A signal operation removes one process from the list of waiting processes and awakens that process.

The wait semaphore operation can now be defined as void wait(semaphore s)

```
{
    s.value....;
    if(s.value < 0)
    { add this process to s.L;
      block( ) ;
    }
}
```

The signal semaphore operation can now be defined as void signal (semaphore s)

```
{
    s.value++;
    if(s.value < = 0) {
    remove a process P from s.L;
    wakeup(P);
}
```

The block operation suspends the process that invokes it. The wakeup(P) operation resumes the execution of a blocked process P. These two operations are provided by the operating system as basic system calls.

Solution for Bounded Buffer Producer/Consumer Problem using Semaphores – Refer Q.27.

Q.29. Discuss dining philosophers problem. Also give a solution to it.
(R.G.P.V., Dec. 2005)

Ans. There is a rounded table. Five philosophers are seated around this table to take food. Each philosopher has a plate of food. Each philosopher needs two spoons to eat food. There is only one spoon between each pair of plates. This situation is shown in fig. 4.18.

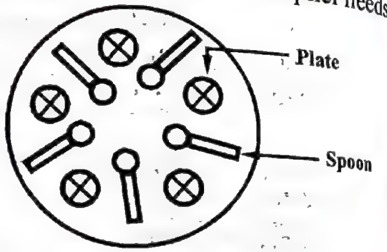


Fig. 4.18 Structure of Dining Table in Philosopher Problem

Each philosopher can eat or think. When a philosopher wants to eat food, he tries to take his left and right spoons, one at a time, in any order. If philosopher found both spoons, he starts eating for a while and then puts down the spoons and starts thinking again.

Solutions of Dining Philosophers Problem –

(i) First Solution – Fig. 4.19 shows this solution. The procedure take_spoon waits until the specified spoon is available and the seizes it.

This solution is not always correct. If all five philosophers become hungry simultaneously and each take his left spoon then there is no right spoon available in their right side. It means there will be a deadlock.

```
# include "prototypes.h"
# define N 5              /* number of philosophers*/
void philosopher (int i) /* i : which philosopher (0 to N –1) */
{
    while (true)
    {
        think ( );/* philosopher is thinking */
        take_spoon(i);         /* take left spoon */
        take_spoon ((i + 1)%N); /* take right spoon */
        eat ( );
        put_spoon(i); /* put left spoon back on the table */
        put_fork ((i + 1)%N); /* put right spoon back on the table */
    }
}
```

Fig. 4.19 A Nonsolution to the Dining Philosophers Problem

Since this program suffers from deadlock problem, it can be modified so that after taking the left spoon, the program checks to see if the right spoon is available. If it is not, the philosopher puts down the left spoon, waits for some time, and then repeats the whole process. Unfortunately, this approach also fails. If all the philosophers start the algorithm simultaneously, picking up their left spoons, then right spoons will not be available, putting down their left spoons, waiting, picking up their left spoons again simultaneously, and so on, forever. Hence, this situation leads to starvation problem.

(ii) Second Solution – The solution is to protect the five statements following the call to think by a binary semaphore so that there is no deadlock and starvation. Before starting to grab spoons, a philosopher will do a DOWN on mutex. After replacing the spoons, he will do an UP on mutex. This solution is adequate from a theoretical viewpoint. From a practical viewpoint, it has a performance bug that only one philosopher can be eating at any instant. With five available spoons, only two philosophers can be allowed to eat at the same time.

```
#include "prototype.h"
#define N 5                  /* number of philosophers */
#define LEFT (i – 1)% N      /* number of i's left neighbour */
#define RIGHT ( i + 1) %N    /* number of i's right neighbour */
#define THINKING 0           /* philosopher is thinking */
#define HUNGRY 1             /* philosopher is trying to grab spoons */
#define EATING  2            /* philosopher is eating */
typedef int semaphore;
int state[N];                /* array to keep track of everyone's state */
semaphore mutex = 1;         /* mutual exclusion for critical regions */
semaphore s[N];              /*one semaphore for each philosopher*/
                             /*  procedure that executed for each
                                 philosopher */
void philosopher_process(int i)  /* i: which philosopher (0 to N – 1) */
{
    while(1)                 /* repeat forever */
    {
        think();             /* philosopher is thinking */
        pickup_spoons(i);    /* grab two spoons or block */
        eat_food();
        put_down_spoons(i);  /* put both spoons back on table */
    }
}
                             /* procedure for picking the spoons */
void pickup_spoons (int i)   /* i: which philosopher (0 to N – 1)*/
{
    down(&mutex);            /* enter critical region */
    state[i] = HUNGRY;       /* record fact that philosopher i is hungry */
    test(i);                 /* try to grab 2 spoons */
    up(&mutex);              /* exit critical section */
    down(&s[i]);             /* block if spoons were not grabbed */
}
                             /* procedure  for putting down the spoons */
void put_down_spoons(int i)  /* i : which philosopher (0 to N – 1) */
{
    down(&mutex) ;
    state[i] = THINKING;     /* philosopher has finished eating */
    test(LEFT);              /* see if left neighbour can now eat */
    test(RIGHT);             /* see if right neighbour can now eat */
    up(&mutex);
}
void test(int i)
{
    if(state[i] = = HUNGRY && state[LEFT] ! = EATING && state[RIGHT]!
                                                         = EATING)
    {
    state[i] = EATING;
}   up(&s[i]); }
```

Fig. 4.20 A Solution to the Dining Philosophers Problem

Q.30. Write a semaphore solution for dining philosopher's problem. **(R.G.P.V., Dec. 2016)**

Ans. Refer Q.29.

Q.31. What is the significance of dining philosopher's problem in the study of O.S. ?

Ans. This problem, due to Dijkstra, may not seem important or relevant in itself. However, it does illustrate basic problems in deadlock and starvation. Furthermore, attempts to develop solutions reveal many of the difficulties in concurrent programming. In addition, the dining philosophers problem can be seen as representative of problems dealing with the coordination of shared resources, which may occur when an application includes concurrent threads of execution. Accordingly, this problem is a standard test case for evaluating approaches to synchronization.

Q.32. State the readers and writers problem.

Ans. Readers and writers is a classical problem in concurrent programming. It revolves around a number of processes using a shared global data structure (such as a file or record). The processes are categorized into readers and writers depending on their usage of the resource. A reader never modifies the shared data structure, whereas a writer may both read and write it. A number of readers may use the shared data structure concurrently, because no matter how they are interleaved, they cannot possibly compromise its consistency. On the other hand, writers must be granted exclusive access to data, i.e., they cannot be interleaved safely with either readers or other writers.

The readers-writers problem has several variations, all involving priorities. The simplest one, referred to as the first readers-writers problem. It requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object. It means that no reader should wait for other readers to finish because a writer is waiting. The second readers-writers problem requires that, once a writer is ready, that writer performs its write as soon as possible. It means that if a writer is waiting to access the object, no new readers may start reading.

A solution to either problem may result in starvation. In the first case, writers may be starve; in the second case, readers may starve.

Q.33. Give the solution to the readers and writers problem.

Ans. The reader processes share the following data structures

```
semaphore mutex, wrt;
int readcount;
```


The semaphores mutex and wrt are initialized to 1 and readcount is initialized to 0. The semaphore wrt is common to both the reader and writer processes. The mutex semaphore is used to ensure mutual exclusion when the variable readcount is updated. The variable readcount keeps track of how many processes are currently reading the object. The semaphore wrt functions as a mutual-exclusion semaphore for the writers. It is also used by the first or last reader that enters or exits the critical section. It is not used by readers who enter or exit while other readers are in their critical sections.

The code for a writer process is shown in fig. 4.21 and the code for a reader process is shown in fig. 4.22.

```
wait(wrt);
.....
      writing is performed
      .....

signal(wrt);
```

Fig. 4.21 The Structure of a Writer Process

```
wait (mutex);
readcount + +;
if (readcount = = 1)
      wait (wrt);
signal (mutex);
      ......
      reading is performed
      .....
wait (mutex);
readcount - - ;
if (readcount = = 0)
      signal (wrt);
      signal (mutex);
```

Fig. 4.22 The Structure of a Reader Process

Q.34. Discuss any one classical problem of synchronization. **(R.G.P.V., June 2017)**

Ans. Refer Q.32.

Q.35. What are monitors ? How are they useful in process synchronization? Discuss the features of it. **(R.G.P.V., Dec. 2011)**

Ans. A monitor is a high-level synchronization construct. A monitor is characterized by a set of programmer-defined operators. The representation of a monitor type consists of declarations of variable whose values define the state of an instance of the type, as well as the bodies of procedures or functions that implement operations on the type. The syntax of a monitor is shown in fig. 4.23.

The monitor construct ensures that only one process at a time can be active within the monitor. Consequently, the programmer does not need to code this synchronization constraint explicitly. However, the monitor construct is not sufficiently powerful for modeling some synchronization schemes. For this purpose, additional synchronization mechanisms are defined. These mechanisms are provided by the condition construct. A programmer who needs to write her own tailor-made synchronization scheme can define one or more variables of type condition –

condition x, y;

The only operations that can be invoked on a condition variable are wait and signal. The operation

x.wait();

means that the process invoking this operation is suspended until another process invokes

x.signal();

The x.signal operation resumes exactly one suspended process. If no process is suspended, the signal operation has no effect. It means that the state of x is as though the operation were never executed.

A monitor supports synchronization through the use of condition variables that are contained within the monitor and accessible only within the monitor. Two functions operate on condition variables are –

```
monitor monitor - name
{
    shared variable declaration.
    procedure body p1        (. . .)    {
    . . . .
    }
    procedure body p2        (. . .)    {
       . . . .
    }
         .
         .
         .
    procedure body pn        (. . .)    {
    }
    {
        initialization code
    }
}
         .
         .
         .
    procedure body pn        (. . .)    {
    }
    {
        initialization code
    }
}
```

Fig. 4.23 Syntax of a Monitor

(i) cwait(c) – Suspend execution of the calling process on condition c. The monitor is now available for use by another process.

(ii) csignal(c) – Resume execution of some process suspended after a cwait on the same condition. If there are several such processes, choose one of them. If there is no such process, do nothing.

The major features of a monitor are as follows –

(i) A process enters the monitor by invoking one of its procedures.

(ii) Local data variables are accessible only through the monitor's procedures.

(iii) In the monitor, only one process may be executing at a time; any other process that has invoked the monitor is suspended, waiting for the monitor to become available.

DEADLOCKS – DEADLOCK PROBLEMS, CHARACTERIZATION, PREVENTION, AVOIDANCE, RECOVERY

Q.36. What is deadlock problem ?

Or

Define deadlock. ***(R.G.P.V., Dec. 2015)***

Ans. A deadlock is a situation where a group of processes are permanently blocked as a result of each process having acquired a subset of the resources needed for its completion and waiting for release of the remaining resources held by others in the same group. Thus, making it impossible for any of the processes to proceed.

Deadlock can occur in concurrent environments as a result of uncontrolled granting of system resources to requesting processes.

Deadlock can be defined formally as follows –

A set of processes is deadlock if each process in the set is waiting for an event that only another process in the set can cause.

Because all the processes are waiting, none of them will ever cause any of the events that can wake up any of the other members of the set. Thus, all the processes continue to wait forever.

Q.37. What difficulties may arise, when a process is rolled back as a result of deadlock ? ***(R.G.P.V., June 2009)***

Ans. The process must either be rolled back completely (i.e., aborted and restarted) or just to a previous safe state. In the first case, the cost of the rollback would be determined by a combination of factors including its priority, how long it had computed and how much longer before finishing, as well as the types of resources requested. In the second case, the system would need to maintain more information about each process in order to determine its last safe state.

Q.38. Consider a system consisting of four resources of the same type that are shared by three processes each of which needs at most two resources. Show that the system is deadlock free. ***(R.G.P.V., June 2009)***

Ans. Assume the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and therefore it will return its resources when done.

Q.39. A system has two processes and three identical resources. Each process needs a maximum of two resources to complete. Is deadlock possible ? Explain your answer. ***(R.G.P.V., June 2016)***

Ans. Assume the program is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are two processes and three resources, one process must be able to obtain two resources. This requires no more resources so it will execute and there after it will return all the held resources i.e., 2. Now, the other process will use 1 resource to finish it's execution and then return all the held resources. Both processes will execute without any problem, so we can say that no deadlock is possible.

Q.40. What are the different methods for deadlock handling?

Ans. There are three methods for deadlock handling –

(i) Ensuring that Deadlock State Will Never Occur – For this purpose two techniques are used – Deadlock prevention and deadlock avoidance.

(a) Deadlock Prevention – In this scheme, deadlock is prevented by ensuring that at least one condition occur out of four conditions (mutual exclusion, hold and wait, no preemption, circular wait) required for occuring deadlock state.

(b) Deadlock Avoidance – This scheme requires additional information about how resources will be requested by a process in its lifetime. By this information, it takes decision whether resource should be allocated to the process or it should wait. Here, all available resources are allocated currently and resources will be requested in future are kept in account.

(ii) After Occurring Deadlock – In this system we allow the system to enter a deadlock state, detect it, and recover. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred, and an algorithm to recover from the deadlock.

(iii) Some systems does not ensure that a deadlock will never occur, and also does not provide a mechanism for deadlock detection and recovery, then we may arrive at a situation where the system is in deadlock state yet has no way of recognizing what has happened. This method is not viable, approach to the deadlock problem, however, in many systems deadlock occur infrequently, once per year, thus, this method is cheaper than the costly deadlock prevention, deadlock avoidance, or deadlock detection and recovery methods that must be used constantly. Also, in some circumstances the system is in a frozen state but not in a deadlock state.

Q.41. What are the characteristics of deadlock ?

Or

What are the conditions necessary to hold for deadlock occur ? ***(R.G.P.V., Dec. 2012)***

Ans. The following conditions are necessary for a deadlock situation to occur in a system –

(i) Mutual-exclusion Condition – If a resource is held by a process, any other process requesting for that resource must wait until the resource has been released.

(ii) No-preemption Condition – A resource that has been allocated to a process becomes available for allocation to another process only after it has been voluntarily released by the process holding it.

(iii) Hold-and-wait Condition – Process are allowed to request for new resources without releasing the resources that they are currently holding.

(iv) Circular-wait Condition – Two or more processes must form a circular chain in which each process is waiting for a resources that is held by the next process of the chain.

All the four conditions must hold simultaneously in a system for a deadlock to occur. If any one of them is absent, no deadlock can occur. The four conditions are not completely independent because the circular-wait condition implies the hold-and-wait condition. Although these four conditions are some what interrelated, it is quite useful to consider them separately to devise method for deadlock prevention.

Q.42. What is deadlock ? What are the four necessary condition for a deadlock to occur ? ***(R.G.P.V., Dec. 2016)***

Ans. Refer Q.36 and Q.41.

Q.43. Write a protocol which ensures that the following conditions never hold in the system –

(i) Mutual exclusion ***(ii) Hold and wait***
(iii) No preemption ***(iv) Circular wait.***

Or

Write short note on deadlock prevention. ***(R.G.P.V., Dec. 2006)***

Ans. Deadlock prevention is an approach used by designers in dealing with the problem of deadlock. The basic philosophy of deadlock prevention is to deny at least one of the four necessary conditions for deadlocks.

(i) Mutual Exclusion – The mutual exclusion condition must hold for nonsharable resources, such as a printer cannot be simultaneously shared by several processes. On the other hand, sharable resources do not require

mutually exclusive access and thus cannot be involved in a deadlock. For example, read only files can be accessed simultaneously by several processes.

To prevent mutual exclusion, avoid assigning a resource when that is not absolutely necessary, and try to make sure that a few processes may claim the resource as possible as.

(ii) Hold and Wait – The hold and wait condition can be eliminated by forcing a process to release all resources held by it whenever it requests a resource that is not available. There are two protocols to implement this strategy –

(a) The process requests all needed resources before it begins execution.

We can implement this provision by requiring that system calls requesting resources for a process precede all other system calls.

(b) Second protocol allows a process to request resources only when the process has none. A process requests resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

These protocols have two main disadvantages –

(a) Resource utilization may be slow, since many of the resources may be allocated but unused for a long period.

(b) Starvation is possible.

(iii) No Preemption – This condition can be prevented using following protocol. First, if a process holding certain resources is denied a further request, that process must release its original resources, if necessary, requests them again together with the additional resource.

Alternatively, if a process requests a resource that is currently held by another process, the operating system may preempt the second process and require it to release its resources and allocate them to the requesting process. This latter scheme prevents deadlock only if no two processes possessed the same priority.

This protocol is practical only when applied to resources whose start can be saved and restored later, such as CPU registers and memory space.

(iv) Circular Wait – The circular wait condition can be prevented by defining a linear ordering of resource types. If a process has been allocated resources of type R, then it may subsequently request only those resources of types following R in the ordering.

A disadvantage of this approach is that resources must be acquired in the prescribed order as opposed to being requested when actually needed. This may cause some resources to be acquired in advance of their use, thus lowering the degree of concurrency by making unused resources unavailable for allocation to other processes.

Q.44. Explain deadlock avoidance.

Ans. Deadlock avoidance allows the three necessary conditions mutual exclusion, hold and wait, no preemption, but makes judicious choices to assure that deadlock point is never reached. It means that avoidance allows more concurrency than prevention.

The basic idea of deadlock avoidance it to grant only those requests for available resources that cannot possibly result in a state of deadlock. This strategy is implemented by having the resource allocator examine the effects of granting a particular request. If granting of the resource cannot lead to deadlock, the resource is granted to the requestor. Otherwise, the requesting process is suspended until such time when its pending request can be safely granted. In order to evaluate the safety of the individual states, deadlock avoidance requires all processes to state their maximum number of resources of each type requirements prior to execution. A deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources, and the maximum demands of the process.

Q.45. Explain resource allocation graph algorithm for deadlock avoidance.

Ans. A variant of the resource allocation graph can be used for deadlock avoidance having a resource allocation system with only one instance of each resource type.

A new type of edge, called a claim edge is introduced in addition to the request and assignment edges. A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request resource R_j at some time in the future and is represented by a dashed line. When process P_i requests resource R_j, the claim edge $P_i \rightarrow R_j$ is converted to a request edge. Similarly, when a resource R_j is released by P_i, the assignment edge $R_j \rightarrow P_i$ is converted to a claim edge $P_i \rightarrow R_j$. The resources must be claimed a priori in the system. That is, before process P_i starts execution, all its claim edges must already appear in the resource allocation graph. This condition can be relaxed by allowing a claim edge $P_i \rightarrow R_j$ to be added to the graph only if all the edges associated with process P_i are claim edges.

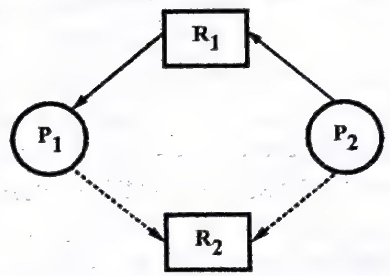


Fig. 4.24 Resource Allocation Graph for Deadlock Avoidance

If a process P_i requests resource R_j. The request can be granted only if converting the request edge $P_i \rightarrow R_j$ to an assignment edge $R_j \rightarrow P_i$ does not result in the formation

of a cycle in the resource allocation graph. If no cycle exist, then the allocation of the resource will leave the system in a safe state. If a cycle exists, then the allocation will put the system in an unsafe state. Therefore, process P_i will have to wait for its requests to be satisfied.

Fig. 4.24 shows a resource allocation graph. If process P_2 requests R_2. This allocation can create a cycle in the graph leading to an unsafe state. If P_1 requests R_2, and P_2 requests R_1, then a deadlock will occur.

Q.46. Explain the resource allocation graph algorithm for deadlock detection with relevant diagrams. ***(R.G.P.V., June 2017)***

Ans. To model the state of resource allocations and requests, a resource alocation graph may be used. In resource allocation graph, resources R_1, R_2, R_3 .... etc. are respresented by rectangles and process P_1, P_2, P_3 .... etc. by circles. A direct arrow is made from process to resource rectangle for every pending resource request, whereas a direct arrow is made from a resource dot to the process for every granted request. Fig. 4.25 shows the resource allocation graph for the system shown in table 4.1.

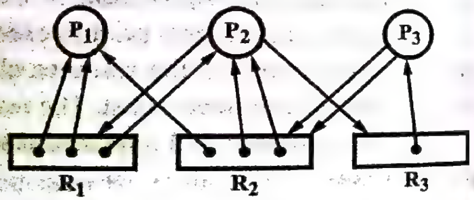


Fig. 4.25 Resource Allocation Graph

Table 4.1

Process	*Current Allocation*			*Outstanding Requests*			*Resource Available*		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	1	0	0	0	0			
P_2	1	2	0	1	0	1	0	0	0
P_3	0	0	1	0	2	0			

A reduced allocation graph is used to compute whether deadlock exists or not. The arrows connected with every process and every resource are checked to reduce a resource allocation graph.

(i) If a resource has only outgoing arrows (means it has no pending request), erase all its arrows.

(ii) If a process has only incoming arrows (means its all requests have been granted), erase all its arrows.

(iii) If a process has outgoing arrows, but for every such request arrow there is an available resource dot (a dot without any leading arrow from it) in the resource the arrow points to, erase all the process's arrows.

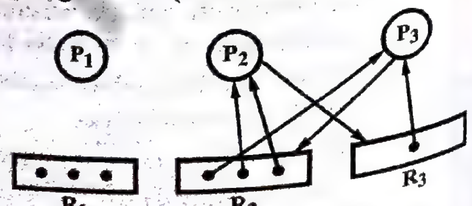


Fig. 4.26 Reduced Resource Allocation Graph

In checking all the processes if any process was found whose arrows could be erased, redo all three procedures. Continue until there are either no process can have its arrow removed or no arrow remaining. If any arrow left which can not be removed then the system is in deadlock state.

Q.47. Illustrate using suitable example, deadlock avoidance with the help of Banker's algorithm. ***(R.G.P.V., Dec. 2006)***

Or

Explain Banker's algorithm for deadlock avoidance with examples. ***(R.G.P.V., June 2016)***

Or

Write Banker's algorithm. ***(R.G.P.V., May 2018)***

Ans. If the necessary conditions for a deadlock to occur are in free place, it is still possible to avoid deadlock by being careful when resources are allocated. Perhaps the most famous deadlock avoidance algorithm is Dijkstra's Bankers' Algorithm, called by this interesting name because it involves a banker who makes loans and receives payments from a given source of capital. This algorithm is used for deadlock avoidance. The statement and assumptions of Banker's algorithm are given below.

Assumptions for Banker's algorithm are as follows –

(i) Every process tells in advance, the number of resources of each type it may require.

(ii) No process asks for more resources than what the system has.

(iii) At the termination time every process will release resources.

Statement – Resources for processes are allocated so that transition is always from one safe state to another state.

According to this algorithm when a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated, otherwise the process must wait until some processes releases enough resources.

Some data structures must be maintained to implement the Banker's algorithm. They are shown below –

Data	*Type*	*Description*	*Example Structure*
Available	vector of m length	Number of available resources of each type	Available [β] = γ, γ instance of resource type R_β are available.

Max	n×m matrix	Maximum number of resources of each type demanded by each process.	Max $[\alpha,\beta] = \gamma$, process P_α may require at most γ instance of resource type R_β.
Allocation	n×m matrix	Number of resources of each type currently allocated to each process.	Allocation $[\alpha,\beta] = \gamma$, process P_α is currently allocated γ instance of resource type R_β.
Need	n×m matrix	Remaining required resources of each process.	Need$[\alpha,\beta] = \gamma$, process P_α may require γ resource of R_β type.

Where n is the number of processes and m is the number of resources. These data structures are vary over time in both size and value.

Each row in the matrices ***Allocation*** and ***Need*** are vectors and refer to them as ***Allocation$_i$*** and ***Need$_i$*** respectively. The vector ***Allocation$_i$*** specified the resources currently allocated to process ***P_i*** the vector ***Need$_i$*** specifies the additional resources that process ***P_i*** may still request to complete its task.

Safety Algorithm – The algorithm for finding out whether or not a system is in a safe state can be described as follows –

(i) Let ***Work*** and ***Finish*** be vectors of length m and n, respectively.
Initialise Work = Available and
Finish[i] = false for i = 1, 2, ..., n.

(ii) Find an i such that
Finish[i] = false, and
$Need_i \le Work$
If no such i exists, go to step (iv).

(iii) Work = Work + $Allocation_i$
Finish[i] = true
go to step 2

(iv) If Finish[i] = true for all i, then the system is in a safe state.

This algorithm requires an order of $m \times n^2$ operations.

Resource-request Algorithm – To satisfy the request made by process P_i for resource following action are required –

(i) If $Request_i \le Need_i$, go to step (ii)
else display error message

(ii) If $Request_i \le Available$, go to step (iii)
else P_i must wait until the resource available.

(iii) Available = Available – $Request_i$
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i - Request_i$

If the resulting resource allocation state is in safe state then sources are allocated to P_i.

Example – Consider a system with five processes P_0 through P_4 and three resource types A, B, C. Resource type A has 10 instances, resource type B has 5 instances, and resource type C has 7 instances. Now consider the situation given below at time T_0.

Process	*Allocation*			*Max*			*Available*		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

The content of the matrix Need is defined to be Max – Allocation and is

Process	*Need*		
	A	*B*	*C*
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

The sequence $< P_1, P_3, P_4, P_2, P_0 >$ satisfies the safety criteria, thus the system is in safe sate. Now suppose process P_1 requests one additional instance of resource type A and two instances of resource type C, so $Request_1 = (1, 0, 2)$. Whether this request can be immediately granted, we first check that $Request_1 \le Available$ [that is, $(1, 0, 2) \le (3, 3, 2)$], which is true, We arrive at the following new state –

Process	***Allocation***			***Need***			***Available***		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

To determine whether the new system is in safe state or not, we execute our safety algorithm and find that the sequence $<P_1, P_3, P_4, P_0, P_2>$ satisfies our safety requirement. So we can immediately grant the request of process P_1.

Q.48. What are the various ways to avoid deadlock ? ***(R.G.P.V., Dec. 2011)***

Ans. Refer Q.45 and Q.47.

Q.49. What is deadlock detection and recovery technique? Discuss some detection methods and recovery handling procedures.

(R.G.P.V., June 2005, Dec.2014)

Or

Describe about how recovery from deadlock. ***(R.G.P.V., Dec. 2015)***

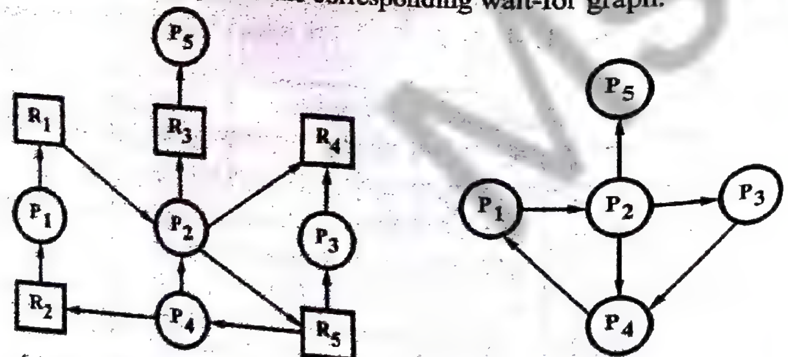
Or

Explain any one deadlock detection methods. ***(R.G.P.V., May 2018)***

Ans. If a system does not employ either a deadlock prevention or a deadlock avoidance algorithm, a deadlock situation may occur. Another technique to deal with deadlocks is detection and recovery. When this technique is used the system does not attempt to prevent deadlocks from occurring. Instead, it lets them occur, tries to detect when this happens, and then takes some action to recover after the deadlock.

(i) Deadlock Detection with One Resource of Each Type – For such a system, a deadlock detection algorithm that uses a variant of the resource – allocation graph, called a ***wait-for*** graph. An edge from P_i to P_j in a wait-for graph implies that process P_i is waiting for process P_j to release a resource that P_i needs. An edge $P_i \rightarrow P_j$ exists in a wait-for graph if and only if the corresponding resource allocation graph contains two edges $P_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for resource R_q.

If this graph contains one or more cycles, a deadlock exists. Any process that is part of a cycle is deadlocked. If no cycles exist, the system is not deadlocked. An algorithm to detect a cycle in a graph requires an order of n^2 operations where n is the number of vertices in the graph. Fig. 4.27 shows a resource allocation graph and the corresponding wait-for graph.



(a) Resource Allocation Graph ***(b) Corresponding Wait-for Graph***

Fig. 4.27

(ii) Deadlock Detection with Multiple Resources of Each Type – When multiple copies of some of resources exist, a different approach is used to detect deadlocks. The deadlock detection algorithm in this case, employs following time-varying data structures.

(a) Available – A vector of length m indicates the number of available resources of each type.

(b) Allocation – A n × m matrix defines the number of resources of each type currently allocated to each process.

(c) Request – An n × m matrix indicates the current request of each process. If Request [i, j] = k, then process P_i is requesting k more instances of resource type R_j.

Deadlock Detection Algorithm –

(a) Let Work and Finish be vectors of length m and n respectively.

Initialize Work : = Available.

If $Allocation_i \neq 0$ for i = 1, 2, ..., n, then Finish [i] : = false; otherwise Finish [i] : = true.

(b) Find an index i such that both

1. Finish [i] = false

2. $Request_i \leq Work$

If no such i exists, go to step (d).

(c) Work : = Work + $Allocation_i$

Finish [i] : = true

goto step (b).

(d) If Finish [i] = false, for some i, $1 \leq i \leq n$, then the system is in a deadlock state. Moreover, if Finish [i] = false, then process P_i is deadlocked.

This algorithm requires an order of $m \times n^2$ operations to detect whether the system is in a deadlock state.

Once deadlock has been detected, some strategy is needed for recovery. The various approaches of recovering from deadlock are –

(i) Process Termination – There are two methods to eliminate deadlocks by aborting processes. Both methods require that the system reclaims all resources allocated to the terminated processes.

(a) Abort all Deadlocked Processes – This method will break the deadlock cycle, but at a great expense. These processes may have computed

for a long time, and the results of these partial computations must be discarded and probably recomputed later.

(b) Abort one Process at a Time Until the Deadlock Cycle is Eliminated – This method incurs overhead. Since, after each process is aborted, a deadlock detection algorithm must be invoked to determine whether any processes are still deadlocked.

(ii) Resource Preemption – To eliminate deadlocks using resource preemption, preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken.

This scheme requires three issues to be addressed –

(a) Selecting a Victim – It means that which resources and which processes are to be preempted.

(b) Rollback – If a resource from a process is preempted, what should be done with that process. The process must be rolled back to some safe state, and restart it from that state.

(c) Starvation – It must be guaranteed that resources will not always be preempted from the same process to avoid starvation problem.

NUMERICAL PROBLEMS

Prob.1. Consider the following snapshot of a system –

	Allocation			Max.			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Answer the following questions using Banker's algorithm –

(i) What is the content of the matrix need ?

(ii) If a request from process P_1 arrives for (1, 0, 2) can the request be granted immediately ?

(iii) Is the system in a safe state ?

(R.G.P.V., Dec. 2016)

Sol. Refer Q.47.

Prob.2. Consider the following snapshot of a system –

Process	Allocation				Max.				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	2	1	0	0
P_1	2	0	0	0	2	7	5	0				
P_2	0	0	3	4	6	6	5	6				
P_3	2	3	5	4	4	3	5	6				
P_4	0	3	3	2	0	6	5	2				

Answer the following questions using Banker's algorithm –

(i) Calculate the need matrix

(ii) Is the system currently in a safe or unsafe state ? If safe state then given safe sequence.

(R.G.P.V., June 2010)

Sol. (i) We know that

Need = Max – Allocation

So the need matrix is

Process	Need			
	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	6	6	2	2
P_3	2	0	0	2
P_4	0	3	2	0

(ii) If the system is in a safe state then there must be a safe sequence. To find a safe sequence. We use safety algorithm. The current state is described as follows –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	0	0	2	1	0	0
P_1	2	0	0	0	0	7	5	0				
P_2	0	0	3	4	6	6	2	2				
P_3	2	3	5	4	2	0	0	2				
P_4	0	3	3	2	0	3	2	0				

Here, Need[P_0] ≤ Available, so this request can be satisfied, so the <P_0> initially in sequence therefore following operation will perform.

Available = Available + Allocation [P_0]

and remove the row corresponding to P_0, from allocation and need matrix so that

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	2	0	0	0	0	7	5	0	2	1	1	2
P_2	0	0	3	4	6	6	2	2				
P_3	2	3	5	4	2	0	0	2				
P_4	0	3	3	2	0	3	2	0				

Since, Need[P_3] ≤ Available so we can take P_3 in safe sequence, so the sequence is <P_0, P_3>, therefore the following operation will perform

Available = Available + Allocation [P_3]

and remove the row corresponding to P_3, from allocation and need matrix so that

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	2	0	0	0	0	7	5	0	4	4	6	6
P_2	0	0	3	4	6	6	2	2				
P_4	0	3	3	2	0	3	2	0				

Since need[P_4] ≤ Available, so safe sequence is <P_0, P_3, P_4> and system state

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	2	0	0	0	0	7	5	0	4	7	9	8
P_2	0	0	3	4	6	6	2	2				

Here, Need[P_1] ≤ Available, so safe sequence is <P_0, P_3, P_4, P_1> and system state

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_2	0	0	3	4	6	6	2	2	6	7	9	8

Here, Need[P_2] ≤ Available, so safe sequence is <P_0, P_3, P_4, P_1, P_2>. The system generates safe sequence, so it is in safe state and safe sequence is

<P_0, P_3, P_4, P_4, P_2>

Prob.3. Consider the following snapshot of a system –

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the Banker's algorithm –

(i) What is the content of the matrix need ?

(ii) Is the system in a safe state ?

(iii) If a request from process P_1 arrives for (0, 4, 2, 0), can th request be granted immediately ?

(R.G.P.V., Dec. 2008, 2012

Sol. (i) We know that

Need = Max – Allocation

So the content of the need matrix are –

Process	Need			
	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

(ii) If system is in safe state then there must be a safe sequence. To find a safe sequence, we use safety algorithm. The current state is described as follows –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	0	0	1	5	2	0
P_1	1	0	0	0	0	7	5	0				
P_2	1	3	5	4	1	0	0	2				
P_3	0	6	3	2	0	0	2	0				
P_4	0	0	1	4	0	6	4	2				

Here, Need[P_0] ≤ Available, so this request can be satisfied, so the <P_0> initially in sequence so the following operation will perform

Available = Available + Allocation[P_0]

and remove the row corresponding to P_0, from Allocation and Need matrix. So now

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	1	0	0	0	0	7	5	0	1	5	3	2
P_2	1	3	5	4	1	0	0	2				
P_3	0	6	3	2	0	0	2	0				
P_4	0	0	1	4	0	6	4	2				

Since, Need[P_2] ≤ Available and also

$Need[P_3] \le Available$ so we can take either P_2 or P_3 in safe sequence, let take P_2 in safe sequence so the sequence is $<P_0, P_2>$.

Now after releasing resources system state is as follows –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	1	0	0	0	0	7	5	0	2	8	8	6
P_3	0	6	3	2	0	0	2	0				
P_4	0	0	1	4	0	6	4	2				

Here, $Need[P_1] \le Available$, so safe sequence is $<P_0, P_2, P_1>$ and system state

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_3	0	6	3	2	0	0	2	0	3	8	8	6
P_4	0	0	1	4	0	6	4	2				

Here, $Need[P_4] \le Available$, so safe sequence is $<P_0, P_2, P_1, P_4>$ and system state

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_3	0	6	3	2	0	0	2	0	3	8	9	10

Here, $Need[P_3] \le Available$, so safe sequence is $<P_0, P_2, P_1, P_4, P_3>$. The system generates safe sequence, so it is in safe state.

(iii) Process P_1 gives the request for (0, 4, 2, 0), if this request is satisfied immediately, then system state will be, after following operations –

$$Available := Available - Request[P_1]$$

$$Allocation[P_1] := Allocation[P_1] + Request[P_1]$$

$$Need[P_1] := Need[P_1] - Request[P_1]$$

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	0	0	1	1	0	0
P_1	1	4	2	0	0	3	3	0				
P_2	1	3	5	4	1	0	0	2				
P_3	0	6	3	2	0	0	2	0				
P_4	0	0	1	4	0	6	4	2				

Now, we will determine, whether system is in safe state or not.

Here, $Need[P_0] \le Available$, so this need can be satisfied and following operations are performed –

$$Available = Available + Allocation\ [P_0]$$

and row corresponding to process P_0 is deleted from Allocation and Need matrix both. So the safe sequence is $<P_0>$ and resultant state is –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	1	4	2	0	0	3	3	0	1	1	1	2
P_2	1	3	5	4	1	0	0	2				
P_3	0	6	3	2	0	0	2	0				
P_4	0	0	1	4	0	6	4	2				

Here, $Need[P_2] \le Available$, so safe sequence is $<P_0, P_2>$ and resultant state is –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_1	1	4	2	0	0	3	3	0	2	4	6	6
P_3	0	6	3	2	0	0	2	0				
P_4	0	0	1	4	0	6	4	2				

Here, $Need[P_1] \le Available$ and also $Need[P_3] \le Available$. We can take any of them, let's take P_1 in safe sequence, then safe sequence is $<P_0, P_2, P_1>$ and resultant state –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_3	0	6	3	2	0	0	2	0	3	8	8	6
P_4	0	0	1	4	0	6	4	2				

Here, $Need[P_3] \le Available$ and also $Need[P_4] \le Available$, we take P_3 then safe sequence is $<P_0, P_2, P_1, P_3>$ and resultant state –

Process	Allocation				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_4	0	0	1	4	0	6	4	2	3	14	11	8

Since $Need[P_4] \le Available$ so final safe sequence is $<P_0, P_2, P_1, P_3, P_4>$. Therefore system is in safe state.

So the request for (0, 4, 2, 0) by process P_1, can be granted immediately.

Prob.4. Consider the following snapshot of a system –

	Allocation				Maximum				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using Banker's algorithms –

(i) What is the content of matrix need ?

(ii) Is the system in safe state ?

(R.G.P.V., Dec. 2013)

Sol. (i) Refer Prob.3 (i).

(ii) Refer Prob.3 (ii).

Prob.5. Describe the Banker's algorithm for safe allocation. Consider a system with three processes and three resource types and that time the following snapshot of the system has been taken –

Process	Allocated			Maximum			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8	7	7	10
P_2	2	0	3	4	3	3			
P_3	1	2	4	3	4	4			

(i) Is the current allocation a safe state ?

(ii) Would the following requests be granted in the current safe state ?

(a) Process P_2 requests (1, 0) ***(b) Process P_1 requests (1, 0)***

(R.G.P.V., June 2011)

Sol. For Banker's algorithm, refer Q.47.

(i) We know that

Need = Max – Allocation

So the contents of the need matrix are –

Process	Need		
	R_1	R_2	R_3
P_1	1	4	5
P_2	2	3	0
P_3	2	2	0

If a system is in safe state then there must be a safe sequence. To find a safe sequence, we use safety algorithm. The current state is described as follows –

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	1	4	5	7	7	10
P_2	2	0	3	2	3	0			
P_3	1	2	4	2	2	0			

Here, need$[P_1]$ ≤ Available, so this request can be satisfied, thus, the $<P_1>$ initially in sequence. The following operation will perform –

Available = Available + Allocation$[P_1]$

and remove the row corresponding to P_1 from allocation and need matrix so that

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_2	2	0	3	2	3	0	9	9	13
P_3	1	2	4	2	2	0			

Since Need$[P_2]$ ≤ Available so we can take P_2 in safe sequence, now the sequence is $<P_1, P_2>$, therefore the following operation will perform

Available = Available + Allocation$[P_2]$

and remove the row corresponding to P_2 from allocation and need matrix so that

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_3	1	2	4	2	2	0	11	9	16

Here, Need$[P_3]$ ≤ Available, so safe sequence is $<P_1, P_2, P_3>$. The system generates safe sequence, so it is in safe state and safe sequence is

$<P_1, P_2, P_3>$

(ii) **(a) Process P_2 requests (1, 0)** – This request can be granted since request ≤ available. This request gives the following new state –

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	1	4	5	6	7	10
P_2	3	0	3	1	3	0			
P_3	1	2	4	2	2	0			

Now we must determine, the new state is in safe state or not. To do so, we again execute our safety algorithm and find that the sequence $<P_1, P_2, P_3>$

satisfies our safety requirement. Hence we can immediately grant the request of process P_2.

(b) Process P_1 requests (1, 0) – This request can be granted since request ≤ Available. This request gives the following new state –

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	3	2	3	0	4	5	6	7	10
P_2	2	0	3	2	3	0			
P_3	1	2	4	2	2	0			

Now, we must determine the new state is in safe state or not. To do so, we again execute our safety algorithm and find that the sequence <P_1, P_2, P_3> satisfies our safety requirement. Hence, we can immediately grant the request of process P_1.

Prob.6. Assume a maximum claim reusable resource system with four processes and three resource types. The claim matrix is given by –

$$C = \begin{bmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 5 & 7 & 13 \\ 1 & 1 & 6 \end{bmatrix}$$

where C(i, j) denotes maximum claim of process i for resource j. The total units of each resource type are given by vector (5, 8, 16). The allocation of resources is given by the matrix –

$$A = \begin{bmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{bmatrix}$$

where A(i, j) denotes the number of the units of resource j that are currently allocated to process i –

(i) Find if the current state of the system is safe.

(ii) Find if granting of a request by process 1 for 1 unit of resource type 1 can safely be done.

(iii) Find if the granting of a request by process 3 for 6 units of resources 3 can safely be done.

(R.G.P.V., Dec. 2011)

Sol. The system has four processes P_1 to P_4 and three resources types R_1, R_2, R_3. The given vector and matrices are as follows –

Resource V = (5, 8, 16)

$$\text{Claim} = \begin{bmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 5 & 7 & 13 \\ 1 & 1 & 6 \end{bmatrix}$$

$$\text{Allocation} = \begin{bmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{bmatrix}$$

(i) The current state of the system can be represented as follows –

	Allocation			Max			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	4	4	1	4	4	0	0	1	5	7
P_2	2	0	1	3	1	4	1	1	3			
P_3	1	2	1	5	7	13	4	5	12			
P_4	1	0	3	1	1	6	0	1	3			

If a system is in safe state then there must be a safe sequence. To find a safe sequence, we use safety algorithm.

Since, Need[P_2] ≤ Available and also Need[P_4] ≤ Available so we can take either P_2 or P_4 in safe sequence, let's take P_2 in safe sequence so the sequence is <P_2>. Therefore, the following operation will perform

Available = Available + Allocation [P_2]

and remove the row corresponding to P_2 so that

Process	Allocation			Max			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	4	4	1	4	4	0	0	3	5	8
P_3	1	2	1	5	7	13	4	5	12			
P_4	1	0	3	1	1	6	0	1	3			

Since Need[P_4] ≤ Available so we can take P_4 in safe sequence, so the sequence is <P_2, P_4>. Therefore, the following operation will perform

Available = Available + Allocation [P_4]

and remove the row corresponding to P_4 so that

Process	Allocation			Max			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	4	4	1	4	4	0	0	4	5	11
P_3	1	2	1	5	7	13	4	5	12			

Since Need[P_1] ≤ Available so we can take P_1 in safe-sequence, so the sequence is <P_2, P_4, P_1>. Therefore, the following operation will perform

$$\text{Available} = \text{Available} + \text{Allocation}\ [P_1]$$

and remove the row corresponding to P_1 so that

Process	Allocation			Max			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_3	1	2	1	5	7	13	4	5	12	4	6	15

Here, Need[P_3] ≤ Available, so safe sequence is <P_2, P_4, P_1, P_3>. The system generates safe sequence, so it is in safe sate and safe sequence is

<P_2, P_4, P_1, P_3>

(ii) Now, the process P_1 requests 1 unit of resource 1. To decide that this request can be immediately granted, we first check that request ≤ Available [i.e., (1, 0, 0) ≤ (1, 5, 7)] which is true. Now, suppose that this request has been fulfilled and the new state is as follows –

Now we must determine the new state is in safe state or not. To do so, we again execute our safety algorithm and find that the sequence <P_4, P_2, P_1, P_3> satisfies our safety requirement. Hence, we can safely grant the request of process P_1.

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	1	1	4	3	0	0	0	5	7
P_2	2	0	1	1	1	3			
P_3	1	2	1	4	5	12			
P_4	1	0	3	0	1	3			

(iii) Now, the process P_3 requests 6 units of resource 3, so request$_3$ = (0, 0, 6). To decide that this request can be immediately granted, we first check that request$_3$ ≤ Available, [i.e., (0, 0, 6) ≤ (1, 5, 7)], which is true. Now, suppose that this request has been fulfilled, and the new state is as follows –

Process	Allocation			Need			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	4	4	0	0	1	5	1
P_2	2	0	1	1	1	3			
P_3	1	2	7	4	5	6			
P_4	1	0	3	0	1	3			

Now, we determine, whether system is in safe state or not. Since, there exists no process whose need is less than or equal to available. So, there is no any safe sequences. Hence, the system is now in unsafe state.

⌘⌘

UNIT 5

INTRODUCTION TO NETWORK, DISTRIBUTED AND MULTIPROCESSOR OPERATING SYSTEMS

Q.1. What do you understand by network ? Explain some different types of networks.

Ans. Networks are everywhere, or so it seems. You can hardly do anything with data that does not involve a network. Like the human networks that we are all part of computer networks let us share information and resources. In business, the reliance on networks is even more pervasive than in homes or schools. Networks help individuals and businesses alike save money, but they also help create income. Without a doubt, networking within the home will catch on over the next few years as it has in business. Soon, nearly all individuals in even moderately developed nations will have networked components throughout their homes. Those that don't will be netologically disadvantaged because they will not be able to learn or to function at the same level as those who are networked.

Some of the different types of networks are as follows –

(i) ***Family Network*** – Most people belong to a family network in which related people share their resources and information. This sharing is bi-directional because even the youngest family members share information of some sort. As the family grows, so does the network. Family network is shown in fig. 5.1.

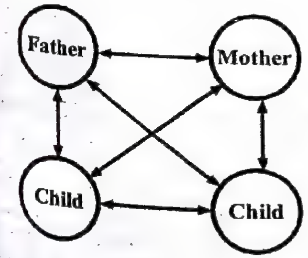


Fig. 5.1

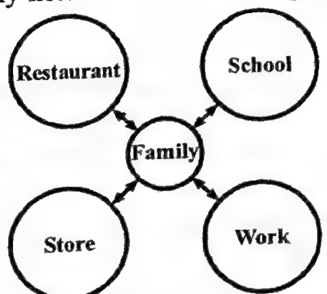


Fig. 5.2

(ii) ***Peer Network*** – Outside the family, there is a community that offers a wider array of resources than the typical family can provide. Naturally,

it makes sense to connect the family to this community to take advantage of the wealth of resources available around town. This type of information/resource sharing can be as simple as loaning a hammer to a neighbour, car-pooling with work associates, or helping a friend with his or her homework. All of these activities involve sharing, or trading, resources. This kind of network is represented by a two-way relationship, a give and take among equals or peers. Peer network is shown in fig. 5.2

(iii) Restaurant Network – The Client and the Server – So, in any type of human network, there's a lot of giving and taking. We already more accustomed to the client/server perspective in networking than we realize. For instance, when we go to dinner at a restaurant, we become customers, or clients, enjoying the food and drink prepared and served to us by the restaurant. On the other hand, the waiter works as a server, controlling and providing his customers with access to resources in the form of placing orders for and delivering food items.

(iv) Computer Networks – A computer network consists of two or more computing devices that are connected in order to share the components of your network (its resources) and the information you store there, as shown in fig. 5.3. The most basic computer network (which consists of just two connected computers) can expand and become more usable when additional computers join and add their resources to those being shared.

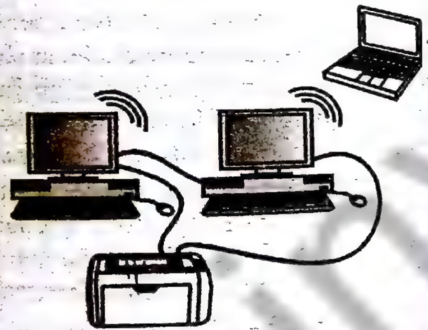


Fig. 5.3

Q.2. What do you understand by network operating system ?

Ans. Network operating systems runs on a server and provides the capability to manage data, users, groups, security, applications, and other networking functions. These type of operating systems allows shared access of files, printers, security, applications, and other networking functions over a small private network. One more important aspect of network operating systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections etc. and that's why these computers are popularly known as tightly coupled systems.

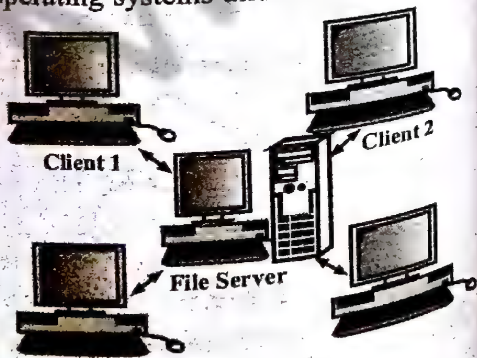


Fig. 5.4

Q.3. Explain the features and functions of network operating system.

Ans. A network operating system (NOS) provides services to clients over a network. Both the client/server and peer-to-peer networking models use network operating systems, and as such, NOSes must be able to handle typical network duties such as the following –

(i) Providing access to remote printers, managing which users are using which printers when, managing how print jobs are queued, and recognizing when devices are not available to the network.

(ii) Enabling and managing access to files on remote systems, and determining who can access what – and who cannot.

(iii) Granting access to remote applications and resources, such as the Internet, and making those resources seem like local resources to the user (the network is ideally transparent to the user).

(iv) Providing routing services, including support for major networking protocols, so that the operating system knows what data to send where.

(v) Monitoring the system and security, so as to provide proper security against viruses, hackers, and data corruption.

(vi) Providing basic network administration utilities (such as SNMP, or simple network management protocol), enabling an administrator to perform tasks involving managing network resources and users.

Q.4. Define NetWare and explain its features.

Ans. NetWare has been a great LAN operating system for years, but only recently (with NetWare 5.x) has NetWare moved beyond the LAN to where it can easily be a part of larger networks. Until quite recently, Novell NetWare used to be the single most used network operating system (NOS). However, first Windows NT, and Windows 2000 and Linux, have steadily eaten into the NetWare market share for network operating systems. Currently, all three operating systems have a roughly equal share of the network operating system market, which means that NetWare is still used in at least one-third of all server systems.

NetWare offers the following features –

(i) Multiprocessor Kernel – This feature enables one NetWare operating system to utilize multiple processors. This process is called symmetric multiprocessing (SMP). SMP enables processors to share memory and bus paths, even coordinating the processing of a single application in parallel.

(ii) NLMs – Where UNIX uses daemons and Windows uses services, NetWare uses NetWare Loadable Modules (or NLMs) to provide services from the server. NLMs are programs that run in the background on the server to provide consistent services to the network.

(iii) PCI Hot Plug – This feature enables administrators to dynamically config ıre PCI network components while the system is running. We can replace, upgrade, or add new cards with the Hot replace, Hot upgrade, and Hot expansion features, respectively.

Q.5. Write down the advantages and disadvantages of network operating system. Also enlist the example of it.

Ans. **Advantages of Network Operating System –**

(i) Highly stable centralized servers.

(ii) Security concerns are handled through servers.

(iii) New technologies and hardware up-gradation are easily integrated to the system.

(iv) Server access are possible remotely from different locations and types of systems.

Disadvantages of Network Operating System –

(i) Servers are costly.

(ii) User has to depend on central location for most operations.

(iii) Maintenance and updates are required regularly.

Examples of Network Operating System – Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD etc. are the examples of network operating system.

Q.6. What are the benefits of computer networking ?

Ans. Computer networking provides following benefits –

(i) It allows us to share data and resources.

(ii) It helps us in reducing the required number of devices.

(iii) It provides us a platform to communicate with other users in network.

(iv) It allows multiple users to work on a single project.

(v) It allows us to store all data in a centralize location.

(vi) It allows us to implement the security policies.

(vii) It allows us to track and monitor the use of resources.

Q.7. Write down the purpose of computer networking.

Ans. The main purpose of computer networking is sharing. In computer networks where we have a lot of things to share, networking is the single best solution. Through networking, we can mainly share three things; data, resources, and applications. Each of these are discussed below –

(i) Data Sharing – Networking allows us to exchange data between computers. Traditional methods of data transportation (such as CD, DVD, USB, etc.) do not work well where data is exchanged frequently.

For example, suppose there are two computers which exchange data regularly. Without networking, the following steps will be required to exchange data between them.

(a) In sender PC, write data in external device (such as CD, DVD and USB).

(b) Take external device to receiver PC.

(c) In receiver PC, read or copy the data from external device.

If data is exchanged 100 times a day, we will have to follow these steps 100 times a day. In such a situation, where the data is exchanged so frequently, this method is neither convenient nor appropriate.

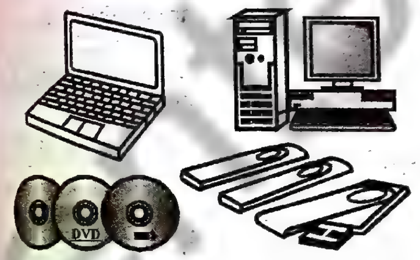


Fig. 5.5

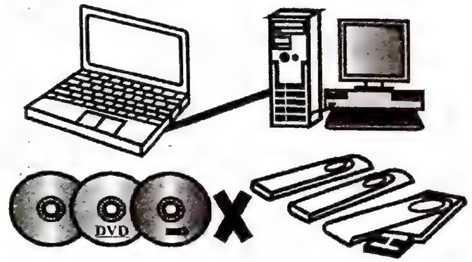


Fig. 5.6

In this case, networking is the single best solution. It allows us to exchange data between connected devices. Once networking is properly setup between both computers, they can exchange data regularly without any external device.

(ii) Resources Sharing – Networking allows us to share devices among the computers. By sharing the devices, we can reduce the number of required components in network.

For example, let's say we have four computers that sometimes require a printer. Without networking, we will have to buy four printers; One for each computer (see fig. 5.7).

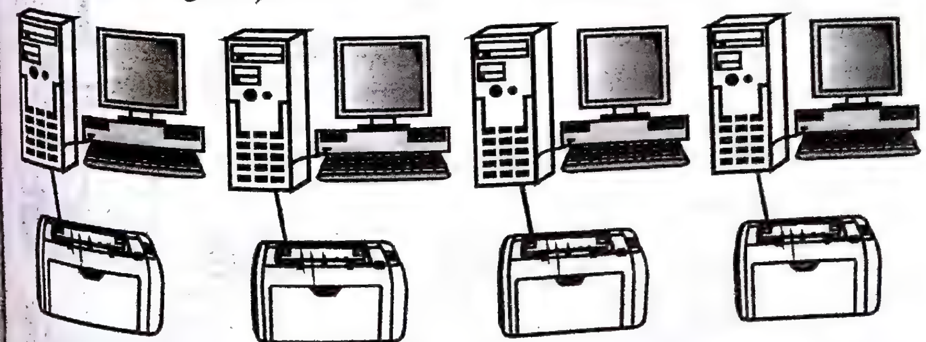


Fig. 5.7

With networking, we only need to buy one printer (see fig. 5.8).

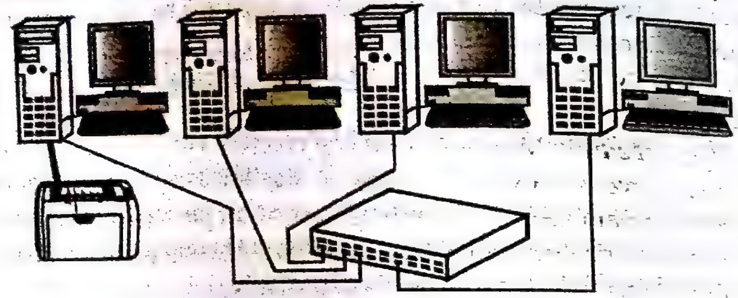


Fig. 5.8

There are two types of device –

(a) Shareable – Devices which can be shared in the network such as hard disk, printer, modem, scanner, CD, DVD, USB etc.

(b) Non-shareable – Devices which cannot be shared in the network such as CPU, RAM, motherboard, monitor, etc.

(iii) Application Sharing – Just like data and resource, through networking we can also share applications. Application sharing is the most common practice in company environment. In companies, usually a project is assigned to several users or a team which have several members. Networking allows concerned users or members to work on assigned project simultaneously.

Q.8. Explain the essential components of computer networking.

Ans. There are four essential components of computer networking –

(i) End devices (ii) Media (iii) Networking devices (iv) Protocol.

(i) End Devices – End device is the device which is used to share and access the shared resource such as computer, laptop, smartphone, etc. For networking, at least two end devices are required. Based on the role played by the end device, it can be categorized in two types – client and server.

(a) Client End Device – End device that is used to access the shared resources such as PC, workstation, smartphone, etc.

(b) Server End Device – End device that is used to share the resource such as PC, server, network printer, etc. For example, fig. 5.9 shows a computer network.

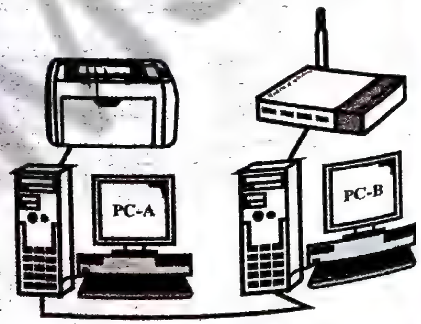


Fig. 5.9

In this network –

(1) When printer is accessed by PC-B, PC-A works as server end device and PC-B works as client end device.

(2) When Internet is accessed by PC-A, PC-B works as server end device and PC-A works as client end device.

This categorization is purely based on the role played by end device in network. A device can be a server device, a client device or both at the same time. The device which is used to access the shared resource is known as client end device while the device which is used to share the resource is known as server end device.

(ii) Media – Media provides the connectivity between end devices. Media is categorized in two types; wired and wireless. In wired media, a cable such as coaxial cable, electrical cable, copper cable or fiber optical cable is used to transmit the data while in wireless media, radio spectrum is used to transmit the data.

Based on several factors (such as speed, distance and signal loose) each media type has its own characteristics and limitations, Different types of network require different type of media.

(iii) Networking Devices – Depending on the size and functionality of a network, several networking devices are used. For example, to build a small home network, we need only few networking devices whereas to build a company network, we may need several hundred networking devices.

(iv) Protocol – When a computer wants to send data to another computer on the network, it starts a session for transmission. During initialization, both computers finalize the transmission rules such as the speed of the transmission, the size of the data file, the safety measurement and flow control for the transmission. These rules are called protocols.

Technically, protocols are the rules that control and define the data transmission process in network. Protocols are defined in various network models such as TCP/IP Layer model, OSI Layer model.

Q.9. Compare network and distributed operating system. ***(R.G.P.V., Dec. 2011)***

Or

Compare and contrast network and distributed operating system. ***(R.G.P.V., Dec. 2013)***

Or

Differentiate between a distributed operating system and a network operating system. ***(R.G.P.V., June 2016)***

Ans. With the era of smaller but powerful computers, distributed processing started becoming a reality. Instead of a centralized large computer, the trend towards having a number of smaller systems at different work sites but connected through a network became stronger.

There were two responses to this development. One was ***Network Operating System (NOS)*** and the other, ***Distributed Operating System (DOS)***. In network operating system, the users are aware that there ar

several computers connected to each other through a network. They also know that there are various databases and files on one or more disks and also the addresses where they reside. But they want to share the data on those disks. Similarly, there is one or more printers shared by various users logged on to different computers. NOVELL's NetWare 286 and the subsequent NetWare 386 operating systems come in this category. In this case, if a user wants to access a database on some other computer, he has to explicitly state its address.

On the other hand, distributed operating system represents a leap forward. It makes the hole network transparent to the users. The databases, files, printers, and other resources are shared amongst a number of users actually working on different machines, but who are not necessarily aware of such sharing. Distributed systems appear to be simple, but they actually are not, oftenly, distributed systems allow parallelisms, i.e. they find out whether a program can be segmented into different tasks which can then be run simultaneously on different machines. On the top of it, the operating system must hide the hardware differences which exist in different computers connected to each other. Normally, distributed systems have to provide for high level of fault tolerance, so that if one computer is down, the operating system could schedule the tasks on the other computers.

Q.10. Contrast centralized and distributed operating system.

(R.G.P.V., Dec. 2013)

Ans. Distributed computing systems are much more complex and difficult to build as compared to traditional centralized systems (those consisting of a single CPU, its memory, peripherals and one or more terminals). The increased complexity is mainly due to effectively using and managing a large number of distributed resources. The system software of a distributed computing system should also be capable of handling the communication and security problems that are very different from those of centralized systems. For instance, the performance and reliability of a distributed system depends to a great extent on the performance and reliability of the underlying communication network. Special software is usually required to handle loss of messages during transmission across the network or to prevent overloading of the network, which degrades the responsiveness and performance to the users. Similarly, special software security measures are required to protect the widely distributed shared resources and services against intentional or accidental violation of access control and privacy constraints.

Q.11. Give the difficulties which can be encounter while implementing a distributed operating system.

(R.G.P.V., Dec. 2015)

Ans. The first problem with the distributed systems is software. With the current state-of-the-art, we do not have much experience in designing, implementing, and using distributed software. This problem will diminish, as more research is done, but for the moment it should not be underestimated.

A second potential problem is because of the communication network. It can lose messages, that needs special software to handle, and it can become overloaded. When the network saturates it must either be replaced or a second one must be added. In both cases, some portion of one or more buildings may have to be rewired at great expense, or network interface boards may have to be replaced. Once the system comes to depend on the network, its loss or saturation can negate most of the advantages the distributed system was built to achieve.

Finally, the easy sharing of data, which is the advantage of distributed systems, may turn out to be a two-edged sword. If people can conveniently access data all over the system, they may equally be able to conveniently access data that they have no business looking at. In other words, security is often a problem.

Q.12. Explain the types of distributed systems.

Ans. The types of distributed systems are as follows –

(i) Client-server – Fig. 5.10 illustrates the simple structure in which client processes interact with individual server processes in separate host computers in order to access the shared resources that they manage. As shown in figure, servers may be clients of other servers. For example, a *search engines*, which enable users to look up summaries of information available on web pages at sites throughout the Internet. These summaries are made by programs called *web crawlers,* which run in the background at a search engine site using HTTP requests to access web servers throughout the Internet. Thus a search engine is both a server and a client – it responds to queries from browser clients and it runs web crawlers that act as clients of other web servers. Here, the server tasks (responding to user queries) and the crawler tasks (making requests to other web servers) are entirely independent; there is little need to synchronize them and they may run concurrently. It fact, a typical search engine would normally include many concurrent threads of execution, some serving its clients and others running web crawlers.

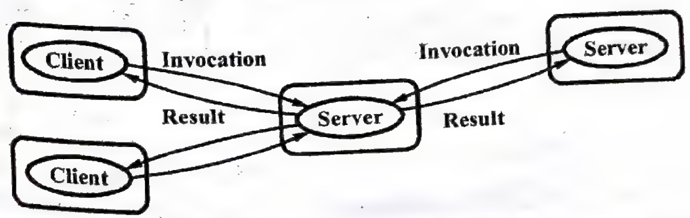


Fig. 5.10 Client Server Architecture

(ii) Peer-to-peer – In this architecture all of the processes involved in a task or activity play similar roles, interacting cooperatively as *peers* without any distinction between client and server processes or the computers that they run on. While the client-server model offers a direct and relatively simple approach to the sharing of data and other resources, it scales poorly. The centralization of service provision and management implied by placing a service at a single address does not scale well beyond the capacity of the computer that hosts the service and the bandwidth of its network connections.

The aim of the peer-to-peer architecture is to exploit the resources in a large number of participating computers for the fulfillment of a given task or activity. Peer-to-peer applications and systems have been successfully constructed that enable tens or hundreds of thousand of computer to provide access to data and other resources that they collectively store and manage. For instance, the Napster application for sharing digital music files. Although it became notorious for reasons other than its architecture, its demonstration of feasibility has resulted in the development of the architectural model in many valuable directions.

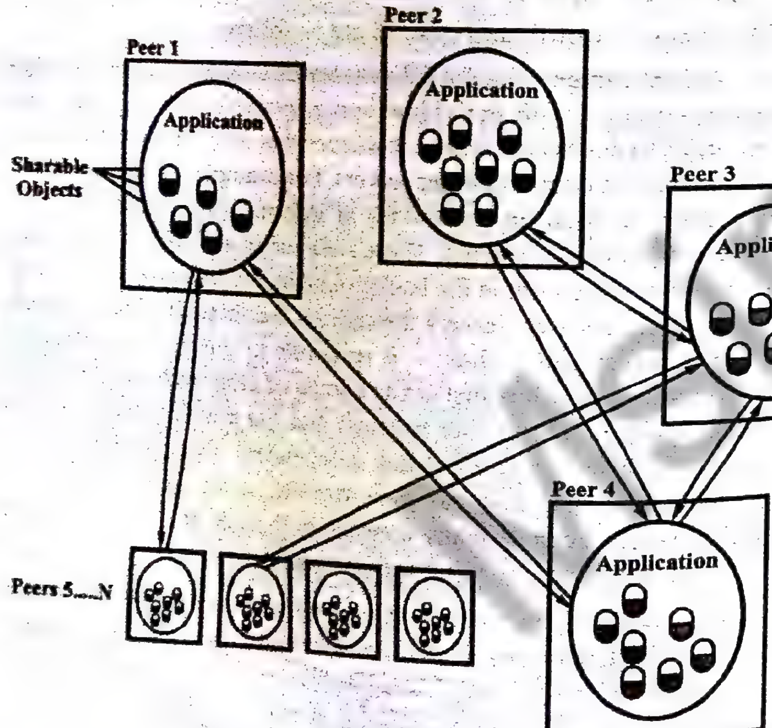


Fig. 5.11 Peer-to-peer Architecture

Fig. 5.11 shows the form of a peer-to-peer application. Applications are composed of large numbers of peers processes running on separate computers

and the pattern of communication between them depends entirely on application requirements. A large number of data objects are shared, an individual computer holds only a small part of the application database and the storage, processing and communication loads for access to objects are distributed across many computers and network links. Each object is replicated in several computers to further distribute the load and to provide resilience in the event of disconnection of individual computers. The need to place individual objects and retrieve them and to maintain replicas amongst many computers renders this architecture substantially more complex than the client server architecture.

Q.13. Describe the distributed computing models.

Ans. Different models used for building distributed computing systems are classified into following categories –

(i) Minicomputer Model – The minicomputer model is a simple extension of the centralized time-sharing system. A distributed system based on this model consists of a few minicomputers interconnected through a communication network. Each minicomputer usually has a number of users simultaneously logged on to it. For this, several interconnective terminals are connected to each minicomputer. Each user is logged on to one specific minicomputer, with remote access to other minicomputers. The network permits a user to remote resources that are available on some machine other than the one on to which the user is currently logged. This model can be used when resource sharing with remote users is desired. An example of a distributed system based on the mini- computer model is ARPANET. Fig. 5.12 shows the minicomputer model.

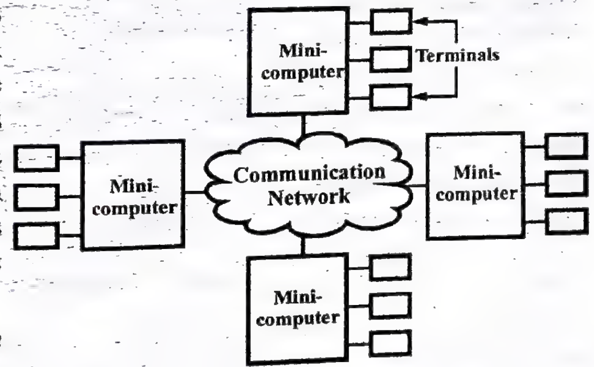


Fig. 5.12 Minicomputer Model

(ii) Workstation Model – A distributed system based on the workstation model consists of various workstations interconnected by a communication network as shown in fig. 5.13. A company's office or a university department may have various workstations scattered

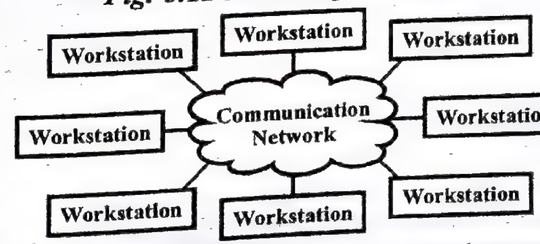


Fig. 5.13 Workstation Model

throughout a building or campus, all workstations equipped with its own disk and serving as a single-user computer, whereas others may be in public areas and have several different users during the course of day.

In this model, a user logs onto one of the workstations called his or her "home" workstation and submits jobs for execution. It transfers one or more of the processes from the user's workstation to some other workstation (currently idle) when the system finds that the user's workstation does not have sufficient processing power for executing the processes of the submitted jobs efficiently and gets the processes executed there, and at last the result of execution is returned to the user's workstation. In this model, the user's processes need not be migrated to the server machines for getting the work done by those machines. This model is difficult to implement.

(iii) Workstation-server Model – This model is a network of personal workstations, each with its own local disk and a local file system. A workstation with its own disk is usually known as ***diskful workstation*** and a workstation without its own disk is known as ***diskless workstation.*** With the proliferation of high-speed networks, diskless workstations have become more popular in network environments as comparison to diskful workstations, thus making the workstation-server popular as comparison to the workstation model for building distributed computing systems. A distributed system based on the workstation server model consists of a few minicomputers and several workstations interconnected through a communication network as shown in fig. 5.14.

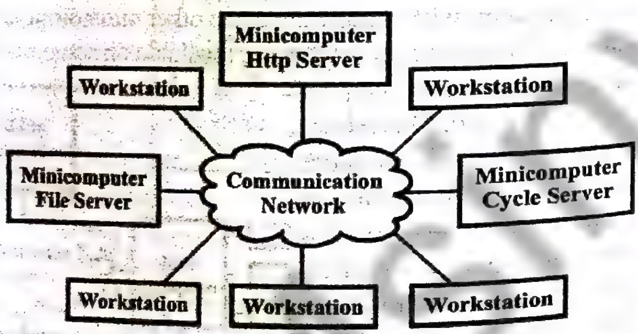


Fig. 5.14 Workstation-server Model

When diskless workstations are used on a network, the file system to be used by these workstations must be implemented either by a diskful workstation or by a minicomputer equipped with a disk for file storage. The minicomputers are used for this purpose. One or more of the minicomputers are used for implementing the file system. Other minicomputers may be used for providing other types of services like print service and database service. Thus, each minicomputer is used as a server machine to provide one or more types of services.

In this model, a user logs onto a workstation called his or her home workstation. Normal computation activities needed by the user's processes are performed at the user's home workstation, but requests for services provided by special servers are sent to a server providing that type of service

that performs the user's requested activity and returns the result of request processing to the user's workstation. Therefore, in this model, the user's processes need not be migrated to the server machines for getting the work done by those machines, For better overall system performance, the local disk of a diskful workstation is normally used for such purposes as storage of unshared files, storage of temporary files, storage of shared files that are rarely changed, caching of remotely accessed data, and paging activity in virtual-memory management.

(iv) Processor-pool Model – This model is based on the observation that most of the time a user does not need any computing power but once in a while he/she may require a huge amount of computing power for a short period of time. Therefore, unlike the workstation-server model where a processor is allocated to each user, in the processor-pool model the processors are pooled together to be shared by the users as required. The pool of processors made up of a huge number of minicomputers and microcomputers attached to the network. Each processor in the pool has its own memory to load and run a system program or an application program of the distributed computing system (see fig. 5.15).

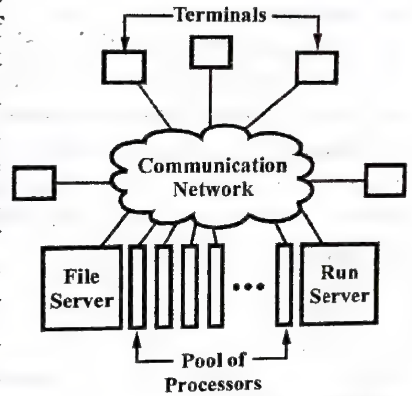


Fig. 5.15 Processor-pool Model

In the processor-pool model, the processors in the pool have no terminals attached directly to them, and users access the system from terminals that are attached to the network by special devices. These terminals are either small diskless workstations or graphic terminals, such as X terminals. A special server allocates and manages the processors in the pool to different users on a demand basis. When a user submits a job for computation, an appropriate number of processors are temporarily assigned to his or her job by the run server. For instance, if the user's computation job is the compilation of a program with n segments where each segment can be compiled independently to produce separate relocatable object files, then to compile all the n segments in parallel n processors from the pool can be assigned to this job. The processors are retruned to the pool for use by other users when the computation is finished. In this model there is no concept of a home machine. That is, a user does not log onto a specific machine but to the system as a whole. This is in contrast to other models where each user has a home machine onto which he or she logs and runs most of his or her progrms there by default.

(v) Hybrid Model – For building distributed system, the workstation-server model is the most widely used model among all the four model. This is

because a large number of computer users only perform simple interactive tasks like editing jobs, sending e-mails, and executing small programs. The workstation-server model is ideal for electronic mails and executing small programs. The workstation-server model is ideal for such simple usage. Although, the processor-pool model is more suitable and performs better in a working environment that has groups of users who often perform jobs needing massive computation.

To combine the advantages of both the workstation-server and processor-pool models a hybrid model may be used to build a distributed system. The hybrid model is based on the workstation-server model but with the addition of a pool processors. The processors in the pool can be assigned dynamically for computations that are too large for workstations or that needs several computers concurrently for efficient execution. Also, in addition to efficient execution of computation-intensive jobs, the hybrid model gives guaranteed response to interactive jobs by allowing them to be processed on local workstations of the users. Although, the hybrid model is more expensive to implement as compared to the workstation-server or processor-pool model.

Q.14. What is client-server computing and explain its terminologies.

Ans. As with other developing phases of the computer field, the client-server computing comes with its new collection of terminologies. As the term depicts, a client-server environment is occupied by clients and servers. The client systems are generally single-user workstations or computers which provide a highly user-friendly interface for the end-user.

The client based station usually presents the type of graphical interface (GUI) which is most comfortable to users that include the requirement of windows and a mouse. Microsoft windows and Mac OS offers an example of those types of interfaces. Client-based applications are customized for simplicity in using and include familiar tools like a spreadsheet. Every server in the client/server environment allows a set of shared services for the clients. The most common category of the server at present used is the database servers.

Terminologies – The terminologies in distributed client/server architecture's are as follows –

(i) Applications Programming Interface (API) – It is a set of task, and it calls programs which allow clients as well as servers for intercommunicating with each other.

(ii) Client/User – It is a networked information requester which is typically a computer system or workstation which can query database and/or other information from a server.

(iii) Middleware – It is a set of drivers, API's and/or other software which improves the connection among a client application with a server.

(iv) Relational Database – It is a type of database wherein the information access becomes limited to the selection of rows which satisfy all search criteria.

(v) Server – It is also a computer, typically a high-powered systems and workstations or a minicomputer or a mainframe, which houses information to manipulate the networked clients.

(vi) Structured Query Language (SQL) – It is a language developed by IBM (International Business Machines) and ANSI had standardized it for addressing, producing, updating and querying RDB's (relational databases).

Q.15. What are the goals of distributed system ? How are concurrency issues tackled in distributed system ? ***(R.G.P.V., June 2010)***

Or

What are the design issues of distributed operating system ? ***(R.G.P.V., Dec. 2011)***

Or

Discuss the issues which need to be looked into in designing a distributed operating system. ***(R.G.P.V., June 2008)***

Or

Explain the design issues of distributed operating system. ***(R.G.P.V., June 2011)***

Or

Write features of distributed operating system. ***(R.G.P.V., June 2016)***

Ans. There are four important goals that should be met to building a distributed system worth the effort. A distributed system should easily connect users to resources, it should hide the fact that the resources are distributed across a network, it should be open, and it should be scalable.

(i) Connecting Users and Resources – The main goal of a distributed system is to make it easy for users to access remote resources, and to share them with other users in a controlled way. Resources can be virtually anything, typical example of resources are printer, storage facilities, data, files, Web pages and network. There are many reasons for wanting to share resources. One reason is economics.

(ii) Transparency – An important goal of distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers. A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be *transparent*.

The concept of transparency can be applied to several aspects of a distributed system as shown in table 5.1.

Table 5.1 Different Forms of Transparency

S.No.	Transparency	Description
(i)	Access	Hide differences in data representation and how a resource is accessed.
(ii)	Location	Hide where a resource is located.
(iii)	Migration	Hide that a resource may move to another location.
(iv)	Relocation	Hide that a resource may be moved to another location while in use.
(v)	Replication	Hide that a resource is replicated.
(vi)	Concurrency	Hide that a resource may be shared by several competitive users.
(vii)	Failure	Hide the failure and recovery of a resource.
(viii)	Persistence	Hide whether a (software) resource is in memory or on disk.

(iii) Openness – Another important goal of distributed systems is openness. An open distributed system is a system that offers services according to standards that describe the syntax and semantics of those services, for example, in computer networks standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols. In distributed systems, services are generally specified through interfaces which are often called an interface definition language (IDL). Interface definition written in an IDL nearly always capture only the syntax of services. They specify precisely the names of the functions that are available together with types of the parameters, return values, possible exceptions that can be raised and so on.

(iv) Scalability – A definite trend in distributed system is toward larger systems. This observation has implication for distributed file system design. Algorithms that work well for systems with 100 machines may work poorly for systems with 1000 machines and not at all for systems with 10,000 machines. For starters, centralized algorithm do not scale well. If opening a file requires contacting a single centralized server to record the fact that the file is open that server will eventually become a bottleneck as the system grows.

(v) Reliability – One of the main goal of building distributed systems was to make them more reliable than single processor systems. The idea is that if some machine goes down, some other machines takes over his jobs. In other words, theoretically the overall system reliability could be the Boolean OR of the component reliabilities. For example, with four file servers, each with a 0.95 chance of being up at any instant, the probability of all four being down simultaneously is $0.05^4 = 0.000006$, so the probability of at least one being available is $(1 - 0.000006) = 0.999994$, far better than any individual server.

(vi) Performance – Building a transparent, flexible, reliable distributed system is worthless if it is as slow as molasses. In particular, when running a particular application on a distributed system it should not be appreciably worse than running that some application on a single processor.

Various performance metrices can be used. Response time is one, but so are throughput, system utilization, and amount of network capacity consumed. Furthermore, the results of any benchmark are often highly dependent on the nature of the benchmark. A benchmark involves a large number of independent highly CPU-bound computations which give radically different results than a benchmark that consists of scanning a single large file for same pattern.

Q.16. What do you mean by distributed operating system ? Discuss the design issues for a distributed operating system. ***(R.G.P.V. Dec. 2012)***

Ans. **Distributed Operating System** – Refer Q.29, Unit-I.

Design Issues – Refer Q.15.

Q.17. Discuss the properties of the following type of operating systems – (i) Interactive (ii) Network (iii) Distributed. ***(R.G.P.V., Dec. 2010)***

Ans. (i) Interactive – It is composed of many short transactions where the results of the next transaction may be unpredictable. Response time need be short (seconds) since the user submits and waits for the result.

(ii) Network – A network operating system provides an environment in which users, who are aware of the multiplicity of machines, can access remote resources by either logging into the appropriate remote machine or transferring data from the remote machine to their own machine.

(iii) Distributed – It distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines, such as a high-speed bus or telephone line.

Q.18. Explain the term multiprocessing. ***(R.G.P.V., Dec. 2008)***

Ans. Refer Q.10, Unit-I.

CASE STUDIES – UNIX/LINUX, WINDOWS AND OTHER CONTEMPORARY OPERATING SYSTEMS

Q.19. Discuss briefly about UNIX.

Ans. Unix is a popular multiuser time-sharing operating system primarily intended for program development and document preparation environments. It was written in a high-level language, C, with careful isolation and confinement

of machine-dependent routines, so that it may be easily ported to different computer systems. As a result, versions of Unix are available for personal computers, microprocessor-based systems, mainframes and supercomputers.

The first version of Unix was written by Ken Thompson, later joined by Dennis Ritchie, at Bell labs in the late sixties. It was a single-user system for PDP-7 computer written in assembly language. After a major rewriting in C and porting to the PDP-11 family of computers, Unix was made available to users outside of AT&T. The main features of Unix are as follows –

(i) Portability (ii) Multiuser operation
(iii) Device independence (iv) Tools and tool-building utilities
(v) Hierarchical file system.

Q.20. What are salient features of UNIX ? ***(R.G.P.V., Dec. 2013)***

Ans. UNIX has different features and they are described as follows –

(i) It is a multiuser and multi-tasking, time-sharing operating system, where each user can execute several tasks simultaneously.

(ii) It is written in high-level language 'C', making it portable to operate on any machine of varying processing power from microprocessors to mainframes.

(iii) It has modularity and allows complex programs to be built from simpler programs.

(iv) It uses hierarchical file structure that allows easy maintenance and efficient implementation.

(v) It hides the machine architecture from the user, making it easier to write programs that run on different hardware implementations.

(vi) It provides good system security.

(vii) It provides the features of I/O redirection and piping.

Q.21. Explain how the features of UNIX operating system supports multitasking environment.

Ans. An UNIX is a popular operating system known for its high reliability, scalability and powerful features. UNIX is known as the backbone of many data centers including the internet.

The capabilities of multitasking, multiuser and portability are the main features of UNIX. Multiple users can access the system with the help of connecting to points known as terminals. Too many users can run multiple programs or processes simultaneously on one system. The UNIX operating system uses a high-level language which is easy to comprehend, modify and transfer to other machines, it means we can change language codes according to the requirements of new hardware on our computer.

Q.22. What is the role of kernel and shell in communication ?

Ans. The kernel is known as the hub of a UNIX operating system and manages the applications and peripherals on a system. The kernel and the shell both, carry out the requests and the commands together. The communication to the system is done through the UNIX shell, which translate to the kernel. A system process starts as the terminal is turn on, that overlooks on the input data. When the password is entered, the system associates the shell program with the terminal. The shell allows to customize options even if the password is technically not saved.

Q.23. Write short note on files and processes in UNIX.

Ans. In UNIX all the functions are involve either a file or a process. Processes can be known as the executions of programs, while files are called as the collection of data created by the user. A files can be in the form of document, programming instructions for the system or a directory. A hierarchical file structure is used by the UNIX in its design, which is started with a root directory signified by the forward slash (/). The root is then followed by its subdirectories, as in an inverted tree and ends with the file.

For example "/Demand/Articles/UNIX.doc". Here the main directory is "Demand" and its subdirectory is "Article", which has a file "UNIX.doc".

Q.24. Explain the execution of processes in Linux/UNIX.

Ans. A program/command when executed, a special instance is provided by the system to the process. This instance consists of all the services/resources that may be utilized by the process under execution.

(i) Whenever a command is issued in UNIX/Linux, it creates/starts a new process. For example, pwd when issued which is used to list the current directory location the user is in, a process starts.

(ii) Through a 5 digit ID number UNIX/Linux keeps account of the processes, this number is call process id or pid. Each process in the system has a unique pid.

(iii) Used up pid's can be ued in again for a newer process since all the possible combinations are used.

(iv) At any point of time, no two processes with the same pid exist in the system because it is the pid that UNIX uses to track each process.

A process can be run in two ways –

(i) Foreground Process – Every process when started runs in foreground by default, receives input from the keyboard and sends output to the screen, when issuing pwd command.

When a command/process is running in the foreground and is taking a lot of time, no other processes can be run or started because the prompt would not be available until the program finishes processing and comes out.

(ii) Background Process – It runs in the background without keyboard input and waits till keyboard input is required. Thus, other process can be done in parallel with the process running in background since they do not have to wait for the previous process to be completed.

Q.25. How does UNIX provide file protection ? Explain.

(R.G.P.V., Dec. 2013)

Ans. There are three types of permissions to a file r(read), w(write), and x(execute). There are three entities to which any combination of these permissions are assigned. These entities are the owner, the group, and the rest (those outside the group). Of the nine characters, the first three characters decide the permissions held by the owner of the file. The next set of three characters specify the permissions for the other users in the group to which the file owner belongs, while the last set decides the permissions for the usesrs outside the group. Out of the three characters belonging to each set the first character is for indicating the *read* permission, the second character is for *write* permission and the last is for *execute* permission.

The permissions for the file *carribeans* from the long listing are rwxr-x--x. Thus, they signify that

(i) The owner can read, write as well as execute the file carribeans.

(ii) The members of the group can read and execute the file, but cannot write to it. A -(hyphen) indicates that the permission is denied.

(iii) All others can only execute carribeans.

These permissions can be encoded numerically. The weights assigned to the three permissions are –

Permission	Weight
read(r)	4
write(w)	2
execute(x)	1

Therefore, when all the permissions are available, the total weightage or value is 4 + 2 + 1 = 7, as is the case with the owner of carribeans.

The group permissions of carribeans are r-x, hence the value is 4 + 0 + 1 = 5. The permissions for the rest are --x, thus the value is 0 + 0 + 1 = 1.

In a nutshell, therefore, we say that carribeans has the permission 751.

The existing file permissions can be changed by the owner of the file or by the superuser. The way to change these permissions is by using *chmod* command. It 'changes the mode' of the file if is executed on. If we want the owner of the carribeans to have all the permissions and the group and others none, we say,

$ chmod 700 myfile

This way of changing file permissions is known as the *absolute mode*. There is another syntax for chmod that changes permissions, which constitutes the symbolic mode. Its general form is –

$ chmod [who] [+/–/=] [permissions] file

The *who* here refers to whom the permissions are to be assigned. It may be the user or owner (u), the group (g) or others (o). If none is specified, all are assumed. The + refers to all permission, – refers to remove permission and = instructs chmod to add the specified permission and take away all others, if present.

The specified permission, of course, can be r, w, or x. The command to give write permission to all will be –

$ chmod +w carribeans

In order to take away execute permission from others as well as group we would say

$ chmod go – x myfile

In order to give a read permission to group and others and take away their write permission for a file called *yankees,* we use chmod as shown below –

$ chmod go + r, go – w yankees

Q.26. Write the description of kernel in UNIX.

Ans. A kernel can be contrasted with a shell (such as bash, csh or ksh in UNIX-like operating systems), which is the outermost part of an operating system and a program that interacts with user commands. The kernel itself does not interact directly with the user, but rather interacts with the shell and other programs as well as with the hardware devices on the system, including, the processor (also called the central processing unit or CPU), memory and disk drives.

The kernel is the first part of the operating system to load into memory during booting (i.e., system startup), and it remains there for the entire duration of the computer session because its services are required continuously. Thus it is important for it to be as small as possible while still providing all the essential services needed by the other parts of the operating system and by the various application programs.

Because of its critical nature, the kernel code is usually loaded into a protected area of memory, which prevents it from being overwritten by other, less frequently used parts of the operating system or by application programs. The kernel performs its tasks, such as executing processes and handling interrupts, in kernel space, whereas everything a user normally does, such as writing text in a text editor or running programs in a GUI (graphical user interface), is done in user space. This separation is made in order to prevent user data and kernel data from interfering with each other and thereby diminishing performance or causing the system to become unstable (and possibly crashing).

When a computer crashes, it actually means the kernel has crashed. If only a single program has crashed but the rest of the system remains in operation,

then the kernel itself has not crashed. A crash is the situation in which a program, either a user application or a part of the operating system, stops performing its expected function(s) and responding to other parts of the system. The program might appear to the user to freeze. If such program is a critical to the operation of the kernel, the entire computer could stall or shut down.

The kernel provides basic services for all other parts of the operating system, typically including memory management, process management, file management and I/O (input/output) management (i.e., accessing the peripheral devices). These services are requested by other parts of the operating system or by application programs through a specified set of program interfaces referred to as system calls.

Q.27. Write short note on Linux operating system.

Ans. Linux is an open-source operating system enhanced and backed by thousands of programmers worldwide. It is a multitasking operating system, which was designed to be used on personal computers. The name "Linux" is derived from its inventor Linus Torvalds.

The Linux operating system is an improved version of UNIX operating system, that has been designed to run as many standard UNIX applications as possible. It is much similar to UNIX operating system but it does not duplicate the basic concept of UNIX. It provides the GUI facility to the UNIX, and all other things look and feel just like UNIX system.

The Linux development is largely around the central operating system kernel the core, that manages all system resources and that interacts directly with the computer hardware. So we need to know basic difference between the Linux kernel and the Linux system. The ***Linux kernel*** is an entirely original piece of software developed from scratch by the Linux community. But the Linux system includes a multitude of components, some written from scratch, some borrowed from other development projects and some created in collaboration with other teams.

The Linux system is a standard environment for application and user programming.

Q.28. Write and explain the features of Linux operating system.

Ans. The features of Linux operating system are as follows –

(i) Fully Protected Multitasking – This means that UNIX can easily switch between tasks without the operating system crashing, because all UNIX processes are separate from those of the operating system. Even if an application crashes, unless it somehow manages to take down the X Windows system with it (which does happen), the operating system just keeps right on humming.

(ii) High Performance and Stability – Many servers running UNIX or Linux have run for years without crashing once. The multitasking capabilities

of UNIX, along with the rapid rate at which the operating system matures (especially with Linux, which is free and can be changed by anyone), make UNIX or Linux a powerful solution, especially for server systems.

(iii) Multiuser Capabilities – True multiuser systems enable different users to be logged into the same system simultaneously. In UNIX and Linux, not only can a user log into the same system at the same time as other users, that user can log in multiple times on the same system as the same user without the operating system batting an eyelash (such things are often necessary when administrating a network, particularly when managing users).

(iv) Tons of High-quality Software – From Apache Server (a web server that's used on a whopping 6 in 10 major web servers on the Internet) to the long-awaited Mozilla.org Mozilla 1.0 open source web browser/e-mail software (Mozilla is an open source version of the venerated Netscape communicator) to the powerful free Gimp graphics manipulation software, Linux is packed with tons of free, high-quality software. The trick is that, with UNIX/Linux, we give up compatibility with commercial software that's available only for Windows and/or Macintosh, currently.

(v) Easy Customization – While other operating systems seem to offer less and less choice to the user about which applications to install with the operating system (Windows XP is this way), UNIX and especially Linux are the exact counterpoint to that model. With UNIX or Linux, we can actually customize our operating system kernel, stripping it down to just drivers and networking or installing everything possible.

(vi) Modular Architecture – The modular architecture of UNIX (and especially Linux) is directly responsible for how customizable UNIX is. Modular really means just what it sounds like – The operating system is built with a kernel that attaches modules to itself based on what the user needs.

(vii) POSIX Compliance – With a free operating system like UNIX, the different distributions (or flavors) of UNIX quickly became difficult to manage. Currently, hundreds of different implementations of UNIX are available. To enable programmers to have some idea of how to code their software such that it would run on any version of UNIX, the Institute of Electrical and Electronics Engineers, Inc. (IEEE) defined the Portable Operaing System Interface (POSIX).

(viii) Use of TCP/IP as the Standard Protocol Stack – UNIX overwhelmingly uses TCP/IP as the protocol stack of choice. If you consider that the vast majority of the servers that help make up the internet are UNIX computers of one form or another, we start to get the idea why TCP/IP is so popular.

Q.29. Explain UNIX/Linux file management system.

Ans. In Unix, there are three basic types of files –

(i) Ordinary Files – An ordinary file is a file on the system that contains data, text, or program instructions.

(ii) Directories – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.

(iii) Special Files – Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable us to access a single file using different names.

Q.30. What are the similarities and differences between Linux and UNIX ?

Ans. The differences between Linux and UNIX are given below –

S.No.	Linux	UNIX
(i)	The source code of Linux is freely available to it's users.	The source code of Unix is not available for the general public.
(ii)	Linux primarily uses graphical user interface with an optional command line interface.	UNIX primarily uses command line interface.
(iii)	Linux OS is portable and can be executed in different hard drives.	UNIX is not portable.
(iv)	Linux is very flexible and can be installed on most of the home based PCs.	UNIX has a rigid environment of the hardware. Hence, cannot be installed on every other machine.
(v)	Linux is used on home based PCs, mobile phones, desktops, etc.	UNIX is mainly used in server systems, mainframes and high end computers.
(vi)	Different version of Linux are Ubuntu, Linux Mint, RedHat, Solaris, etc.	Different versions of UNIX are AIS, HP-UX, BSD, Iris, etc.
(vii)	Linux installation is economical and does not require much specific and high end hardware.	UNIX installation is comparatively costlier as it requires more specific hardware circuitry.
(viii)	The file systems supported by Linux are as follows – xfs, ramfs, nfs, vfat, cramfsm, ext3, ext4, ext2, ext1, ufs, autofs, devpts, ntfs.	The file systems supported by UNIX are as follows – zfs, js, hfx, gps, xfs, vxfs.
(ix)	Linux is developed by an active Linux community worldwide.	UNIX is developed by AT & T developers.

Q.31. Discuss briefly about Windows.

Ans. Microsoft Windows operating system was developed by Microsoft to overcome the limitations of its own MS-DOS operating system. The first successful version of this operating system was Windows 3.0, which was released in 1990. The subsequently released versions were Windows 95, Windows 98 and Windows 2000. The numbers associated with these released versions indicate their year of release. The main features of Windows are as follows –

(i) It provides a graphical user interface (GUI). Hence, it is easier for a new user to learn and use the system.

(ii) It was designed to be not just an operating system, but also a complete operating environment. That is, all its programs conform to a standard way of working.

(iii) It is a single-user, multitasking operating system. That is, a user may run more than one program at a time.

Q.32. Write a brief note on parallel operating system. ***(R.G.P.V., Dec. 2015)***

Or

Write short note on parallel operating system. ***(R.G.P.V., Dec. 2016)***

Ans. Parallel operating systems are the interface between parallel computers (or computer systems) and the applications (parallel or not) that are executed on them. Parallel operating systems are a type of computer processing platform that breaks large tasks into smaller pieces that are done at the same time in different places and by different mechanisms. They are sometimes also described as multicore processors. This type of system is usually very efficient at handling very large files and complex numerical codes. It's most commonly seen in research settings where central server systems are handling a lot of different jobs at once, but can be useful any time multiple computers are doing similar jobs and connecting to shared infrastructures simultaneously. They can be difficult to set up first and can require a bit of expertise, but most technology experts agree that, over the long term, they are much more cost effective and efficient than their single computer counterparts.

A parallel operating system works by dividing set of calculations into smaller parts and distributing them between the machines on a network. To facilitate communication between the processor cores and memory arrays, routing software has to either share its memory by assigning the same address space to all of the networked computers, or distribute its memory by assigning a different address space to each processing core. Sharing memory allows the operating system to run very quickly, but it is usually not as powerful. When using distributed shared memory, processors have access to both their own local memory and the memory of other processors; this distribution may slow the operating system, but it is often more flexible and efficient.

Q.33. Discuss the various aspects to the characterization of a parallel computer operating system.

Ans. The aspects to the characterization of a parallel operating system are as follows –

(i) Coordination – The type of coordination among processors in parallel operating systems is a distinguishing characteristic, which conditions how applications can exploit the available computational nodes. Furthermore, application parallelism and operating system parallelism are two distinct issues– While application concurrency can be obtained through operating system mechanisms or by a higher layer of software, concurrency in the execution of

the operating system is highly dependant on the type of processor coordination imposed by the operating system and the machine's architecture.

(ii) HW vs. SW – As we know that a parallel operating system provides users with an abstract computational model over the computer architecture. It is worthwhile showing that this view can be achieved by the computer's parallel hardware architecture or by a software layer that unifies a network of processors or computers. In fact, there are implementations of every computational model both in hardware or software systems –

(a) The hardware version of the shared memory model is represented by symmetric multiprocessors whereas the software version is achieved by unifying the memory of a set of machines by means of a distributed shared memory layer.

(b) In the case of the distributed memory model there are multicomputer architectures where accesses to local and remote data are explicitly different and as we saw, have different costs. The equivalent software abstractions are explicit message-passing inter-process communication mechanisms and programming languages.

(c) Finally, the SIMD computation model of massively parallel computers is mimicked by software through data parallel programming. Data parallelism is a style of programming geared towards applying parallelism to large data sets, by distributing data over the available processors in a "divide and conquer" mode. An example of a data parallel programming language is HPF (high performance fortran).

(iii) Protection – Parallel computers, being multi-processing environments, require that the operating system provide protection among processes and between processes and the operating system so that erroneous or malicious programs are not able to access resources belonging to other processes. Protection is the access control barrier, which all programs must pass before accessing operating system resources. Dual mode operation is the most common protection mechanism in operating systems. It requires that all operations that interfere with the computer's resources and their management is performed under operating system control in what is called protected or kernel mode (in opposition to unprotected or user mode). The operations that must be performed in kernel mode are made available to applications as an operating system API. A program wishing to enter kernel mode calls one of these system functions via an interruption mechanism, whose hardware implementation varies among different processors. This allows the operating system to verify the validity and authorization of the request and to execute it safely and correctly using kernel functions, which are trusted and well-behaved.

(iv) Distributed Operating System – Refer to Q.10.

⌘⌘⌘

RGPV

B.E. (Fifth Semester) EXAMINATION, June, 2009
(Common for CS, EI & IT Engg.)
OPERATING SYSTEM
(502)

Note : Attempt *one* question from each Unit. All questions carry equal marks. Assume suitable data wherever necessary.

Unit–I

1. (a) What is an operating system ? Discuss the difficulties involved in writing an operating system for a real-time environment. Give examples. (See Unit-I, Page 3, Q.1) 8

(b) Explain the differences in the degree to which the following scheduling algorithms discriminate in favour of short processes – 12

(i) FCFS

(ii) Round Robin

(iii) Multilevel feedback queues.

(See Unit-III, Page 100, Q.23)

Or

2. (a) Describe three circumstances under which blocking I/O should be used. Describe three circumstances under which non-blocking I/O should be used. ** 8

(b) Consider the following jobs – 12

Job	Burst Time
1	10
2	29
3	3
4	7
5	12

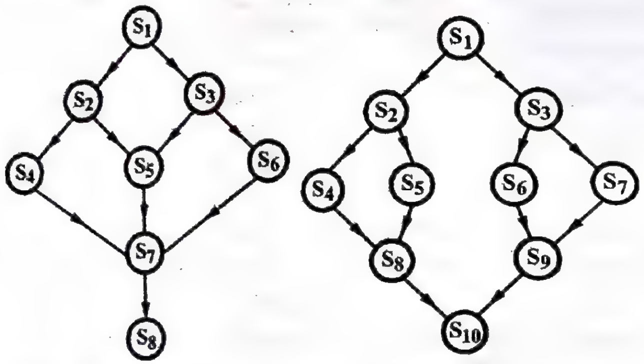
Consider the FCFS, SJF and RR (quantum= 10) scheduling algorithms for this set of processes. Which algorithm would give the minimum average waiting time ? (See Unit-III, Page 113, Prob.7)

Unit-II

3. (a) What difficulties may arise when a process is rolled back as a result of Deadlock ? (See Unit-IV, Page 209, Q.37) 6

(b) What are the necessary conditions for concurrence ? Give a precedence graph. 14

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus



Express the graph using concurrent statement and fork/join construct.

Or

4. (a) Explain scheduling queues with queuing diagram representation of process scheduling. (See Unit-III, Page 98, Q.22) 10

(b) Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock free. 10 (See Unit-IV, Page 209, Q.38)

Unit-III

5. (a) Consider a logical address space of eight pages of 1024 words each, mapped on to a physical memory of 32 frames –

(i) How many bits are in the logical address ?

(ii) How many bits are in the physical address ?

(See Unit-III, Page 156, Prob.10)

(b) Explain paged segmentation with its hardware implementation. 10 (See Unit-III, Page 153, Q.75)

Or

6. (a) Assume memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order). How would each of the first fit, best fit and worst fit algorithms place processes of 212 KB, 417 KB 112 KB and 426 KB. Which algorithm makes the most efficient use of memory ? 8 (See Unit-III, Page 157, Prob.12)

(b) Consider a paging system with the page table stored in memory – 12

(i) If a memory reference takes 1.2 microseconds, how long does a paged memory reference take ?

(ii) If we add 8 associative registers and 75% of all page table reference are found in the associative registers, what is the effective memory reference time ?

(Assume that finding a page table entry in the associative registers takes zero time, if it is there). (See Unit-III, Page 162, Prob.16)

Unit-IV

7. (a) Consider the following page reference strings – 12

1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9

How many page faults would occur for the following replacement algorithms, assuming 2, 4 and 5 frames being made available ?

(i) FIFO (ii) LRU.

(See Unit-III, Page 165, Prob.22)

(b) What is the need to know principle ? Why is it important for a protection system to adhere to this principle ? ** 8

Or

8. (a) What is the cause of thrashing ? How does the system detect thrashing ? Once it detects thrashing, what can the system do to eliminate this problem ? ** 10

(b) Explain trusted system briefly. 10

Unit-V

9. (a) Compare the throughput of C-SCAN and SCAN assuming a uniform distribution of requests. (See Unit-II, Page 74, Q.46) 10

(b) What problems could occur if a system allowed a file system to mounted simultaneously at more than one location ? 10

Or

10. Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For each of three allocation strategies (contiguous, linked and indexed) answer these questions – 20

(i) How is the logical to physical address mapping accomplished in this system ? (For the indexed allocation, assume that a file is always less than 512 blocks long).

(ii) If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk ?

RGPV

B.E. (Fifth Semester) EXAMINATION, June, 2010
(Common for CS & EI Engg.)
OPERATING SYSTEM
CS/EI-502(N)

Note : Attempt any *five* questions.

1. (a) How many types of operating system are there ? Explain each of them. (See Unit-I, Page 20, Q.32) 10

(b) Discuss different structures of operating system with advantages and disadvantages. (See Unit-I, Page 5, Q.7) 10

2. (a) Suppose a disk drive has 300 cylinders, numbered 0 to 299. The current position of the drive is 90. The queue of pending requests in FIFO order is – 10

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

36, 79, 15, 120, 199, 270, 89, 170

Calculate the average cylinder movements for SSTF algorithm.

(See Unit-II, Page 77, Prob.3)

(b) Compare file system in Windows and Linux. 10

(See Unit-II, Page 66, Q.34)

3. (a) Explain Bounded Buffer Problem of synchronization.

(See Unit-IV, Page 201, Q.26)

(b) What is race condition ? How is caused ? What are implications of having a race condition ? **(See Unit-IV, Page 193, Q.16)** 10

4. (a) Consider the following set of processes, with the length of the CPU burst given in milliseconds – 10

Process	Burst Time
P_1	10
P_2	1
P_3	2
P_4	1
P_5	5

Calculate Avg. Turn Around Time and Avg. Waiting Time for Round Robin algorithm. (Assume quantum = 1). **(See Unit-III, Page 117, Prob.9)**

(b) Consider the following snapshot of a system – 10

Process	*Allocation*				*Max.*				*Available*			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	0	0	1	2	0	0	1	2	2	1	0	0
P_1	2	0	0	0	2	7	5	0				
P_2	0	0	3	4	6	6	5	6				
P_3	2	3	5	4	4	3	5	6				
P_4	0	3	3	2	0	6	5	2				

Answer the following questions using Banker's algorithm –

(i) Calculate the need matrix.

(ii) Is the system currently in a safe or unsafe state ? If safe state then given safe sequence.

(See Unit-IV, Page 221, Prob.2)

5. (a) Consider the following page reference string – 10

1, 2, 3, 1, 4, 5, 6, 2, 1, 3, 2, 7, 6, 3, 4, 1, 2, 6

How many page faults would occur for the LRU Replacement algorithm assuming six frames ? All frame are initially empty.

(See Unit-III, Page 163, Prob.19)

(b) Consider a swapping system in which memory consists of the following hole size in memory order 10 kB, 4 kB, 20 kB, 18 kB, 7 kB, 9 kB, 12 kB and 15 kB. Which hole in taken for successive segment requests of –

(i) 12 KB (ii) 10 KB (iii) 9 KB

for first fit, best fit ? **(See Unit-III, Page 161, Prob.14)** 10

(a) What are the goals of distributed system ? How are concurrency issues tackled in distributed system ? **(See Unit-V, Page 245, Q.15)** 10

(b) Explain and compare program threats and system threats. ** 10

(a) Explain Cache memory organization. **(See Unit-III, Page 125, Q.37)** 10

(b) Write comparison between process and thread. 10

(See Unit-III, Page 122, Q.33)

Write short notes on the following – 20

(i) Sensor Network **

(ii) RPC **

(iii) PCB **(See Unit-III, Page 84, Q.3)**

(iv) Device Driver **

(v) System Calls. **(See Unit-I, Page 25, Q.43)**

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2010

(New Scheme)

(Computer Science & Engg. Branch)

OPERATING SYSTEM

[CS/EI-502(N)]

Note : Attempt one question from each Unit. All questions carry equal marks.

UNIT-I

1. (a) Discuss the properties of the following type of operating systems –

(i) Interactive (ii) Network (iii) Distributed.

(See Unit-V, Page 247, Q.17) 10

(b) Describe the evolution of an operating system from simple batch processing to today's operating system with their advantages and disadvantages. **(See Unit-I, Page 20, Q.32)** 10

2. (a) Define essential properties of the following types of operating system – 10

(i) Time sharing **(See Unit-I, Page 12, Q.15)**

(ii) Clustered **(See Unit-I, Page 10, Q.11)**

(iii) Hand held. **(See Unit-I, Page 10, Q.12)**

(b) Describe system call and its types. **(See Unit-I, Page 27, Q.46)** 10

UNIT-II

3. (a) Explain various file allocation methods in detail. 10

(See Unit-II, Page 44, Q.15)

(b) Suppose that a disk has 4000 cylinders numbered 0 to 3999. The drive is currently serving a request at cylinder 143 and the previous request was at 125. The queue of pending request in FIFO order is 86, 1470, 913, 1774, 948, 1500, 1020, 1751, 132. Starting from the current head position, what is the total difference that the disk arm moved to satisfy all the pending request for each of the following disk scheduling algorithm – 10

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(i) FCFS (ii) SCAN (iii) C-SCAN
(iv) SSTF (v) C-Look (vi) Look.

(See Unit-II, Page 79, Prob.5)

4. (a) What are the ways to implement directory ? Explain. 10

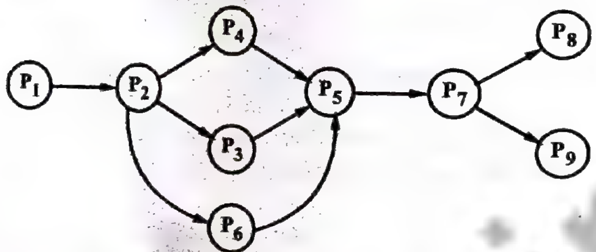
(See Unit-II, Page 49, Q.18)

(b) Discuss the following terms – 10
(i) Memory Protection **
(ii) I/O buffering.

UNIT-III

5. (a) Discuss whether the following primitives and constructs satisfy the bounded wait condition – 10
(i) Semaphores (ii) Critical regions
(iii) Conditional critical region.

(b) Identify transitive procedure relations and independent processes in the following PPG. Also implement using – 10
(i) Fork-Join
(ii) Parabegin-Paraend.



6. (a) Compare and contrast the highest response ratio next scheduling policy with the following policies – 10
(i) Shortest time to go (STG) policy
(ii) Least completed next policy
(iii) Round Robin Policy.

(See Unit-III, Page 103, Q.25)

(b) Explain multilevel adaptive scheduling and fairshare sheduling with examples. (See Unit-III, Page 100, Q.24) 10

unit-iv

7. (a) Discuss Cache memory organization. (See Unit-III, Page 125, Q.37) 10
(b) Find the number of page faults with reference string 0172327103 having 4 page frames and 8 pages. All frames are initially empty. Page replacement strategies are – 10
(i) FIFO (ii) LRU (iii) OPTIMAL.

(See Unit-III, Page 168, Prob.25)

8. (a) Describe the segmented paging scheme of memory management and the hardware required to support the system. 10

(See Unit-III, Page 153, Q.75)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) In a paged segmented system, a virtual address consists of 32 bits of which 12 bits are used for displacement, 11 bits are for segment number and 9 bits are for page number. Calculate – 10
(i) Page size (ii) Max. segment size
(iii) Max. No. of pages (iv) Max. No. of segments.

(See Unit-III, Page 163, Prob.18)

unit-v

9. (a) What are the benefits of a distributed file system when compared to a file system in a centralized system ? ** 10
(b) Discuss system threats and threat monitoring in detail. ** 10

10. Explain the following – 5 each
(i) Parallel processing **
(ii) RMI **
(iii) Distributed shared memory **
(iv) Security design principle. **

RGPV

B.E. (Fifth Semester) EXAMINATION, June, 2011
(Computer Science & Engg. Branch)
OPERATING SYSTEM
[CS–502(N)]

Note : Attempt all questions. All questions carry equal marks.

Unit–I

1. (a) What is Spooling ? What are the advantages of spooling over buffering ? (See Unit-I, Page 31, Q.49) 8
(b) What are real time operating systems ? How are they developed and implemented ? Illustrate some applications where they can be used. (See Unit-I, Page 11, Q.13) 12

Or

Define the essential properties of the following types of OS – 20
(i) Batch (ii) Real time (iii) Time sharing. (See Unit-I, Page 13, Q.16)

Unit–II

2. Write short notes on any *two* of the following – 10 each
(See Unit-II, Page 68, Q.37)
(a) File Access methods **
(b) Interrupt and service routine **
(c) I/O buffering and kernel I/O subsystems.

Unit–III

3. (a) Describe the Banker's Algorithm for safe allocation. Consider a system with three processes and three resource types and at time to the following snapshot of the system has been taken – 15

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Process	Allocated			Maximum			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	2	3	3	6	8	7	7	10
P_2	2	0	3	4	3	3			
P_3	1	2	4	3	4	4			

(i) Is the current allocation a safe state ?

(ii) Would the following requests be granted in the current safe state ?

(1) Process P_2 requests (1, 0) (2) Process P_1 requests (1, 0)

(See Unit-IV, Page 226, Prob.5)

(b) Explain critical section problem. **(See Unit-IV, Page 194, Q.17)** 5

Or

Suppose that the given ahead processes arrive for execution at time indicated – 20

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

Calculate Average turn around time, Average waiting time and throughput –

(i) FCFS (ii) SRTF (iii) Non-Preemptive SJF.

(See Unit-III, Page 107, Prob.2)

Unit–IV

4. Consider the following reference string – 20

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

How many page faults would occur for the following replacement algorithms, assuming three, five or six frames –

(i) LRU (ii) Optimal. **(See Unit-III, Page 168, Prob.26)**

Or

On a system using demand paged memory it takes 200 ns to satisfy a memory request if the page is in memory. If the page is not in memory, the request takes 7 ms if a free frame is availabe or the page to be swapped out has not been modified. It takes 15 ms if the page to be swapped out has been modified. What is the effective access time (E.A.T.) if the page fault rate is 5% and 60% of the time the page to be replaced has been modified ? Assume the system is only running a single process and the CPU is idle during page swaps. **(See Unit-III, Page 178, Prob.31)** 20

5. (a) Explain the design issue of distributed operating system. 10

(See Unit-V, Page 245, Q.15)

(b) Explain the remote procedure call. ** 10

Or

(a) Explain parallel processing and concurrent processing. ** 10

(b) Explain the security aspect of operating system. ** 10

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec., 2011
(Computer Science Engg. Branch)
OPERATING SYSTEM
(CS–502)

Note : Attempt one question from each Unit. All questions carry equal marks.

Unit–I

1. (a) What are the major differences between the following types of operating systems ? 10

(i) Batch system (ii) Real time system (iii) Time sharing system.

(See Unit-I, Page 13, Q.16)

(b) What are the different services provided by an operating system ? Explain each in brief. **(See Unit-I, Page 22, Q.36)** 10

2. (a) Compare the following – 10

(i) Monolithic and layered operating system **(See Unit-I, Page 9, Q.8)**

(ii) Buffering and spooling **(See Unit-I, Page 29, Q.48)**

(iii) Network and Distributed Operating System.

(See Unit-V, Page 237, Q.9)

(b) What are the main functions of an operating system ? 10

(See Unit-I, Page 4, Q.5)

Unit–II

3. (a) Discuss various file access methods. **(See Unit-II, Page 68, Q.37)** 10

(b) Suppose that a disk has 5000 cylinders. The drive is currently serving a request at cylinder 143 and the previous request was at cylinder 125. The queue of pending request in FIFO orders is 86, 1470, 913, 1774, 948, 1509, 1022, 1750 and 130. What is the total distance that the disk arm moves for the following algorithms ? 10

(i) FCFS (ii) SSTF (iii) LOOK (iv) C-SCAN.

(See Unit-II, Page 76, Prob.2)

4. (a) Explain short-term, medium-term and long-term scheduling. 10

(See Unit-III, Page 86, Q.7)

(b) Write a detailed note on interleaving and authentication parameter in file system. ** 10

Unit–III

5. (a) Explain the following terms with examples – 10

(i) Critical section **(See Unit-IV, Page 194, Q.17)**

(ii) Mutual exclusion **(See Unit-IV, Page 188, Q.11)**

(iii) Race condition. **(See Unit-IV, Page 193, Q.16)**

(b) What are monitors ? How are they useful in process synchronization? Discuss the features of it. **(See Unit-IV, Page 207, Q.35)** 10

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

6. (a) What are various ways to avoid deadlock ? 10
(See Unit-IV, Page 218, Q.48)

(b) Assume a maximum claim reusable resource system with four processes and three resource types. The claim matrix is given by – 10

$$C = \begin{bmatrix} 4 & 1 & 4 \\ 3 & 1 & 4 \\ 5 & 7 & 13 \\ 1 & 1 & 6 \end{bmatrix}$$

where C(i, j) denotes maximum claim of process i for resource j. The total units of each resource type are given by vector (5, 8, 16). The allocation of resources is given by the matrix –

$$A = \begin{bmatrix} 0 & 1 & 4 \\ 2 & 0 & 1 \\ 1 & 2 & 1 \\ 1 & 0 & 3 \end{bmatrix}$$

where A(i, j) denotes the number of the unit of resource j that are currently allocated to process i –

(i) Find if the current state of the system is safe.
(ii) Find if granting of a request by process 1 for 1 unit of resource type 1 can safely be done.
(iii) Find if the granting of a request by process 3 for 6 units of resources 3 can safely be done.

(See Unit-IV, Page 228, Prob.6)

Unit-IV

7. (a) Explain the difference between the following – 10
(i) MVT and MFT (See Unit-III, Page 128, Q.41)
(ii) Internal and external fragmentation (See Unit-III, Page 132, Q.44)
(iii) Logical and physical address. (See Unit-III, Page 127, Q.39)

(b) Consider the following page reference string – 10
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6
How many page faults will occur for LRU, FIFO and optimal page replacement algorithms, assuming 4 available frames (initially all empty) ? (See Unit-III, Page 170, Prob.27)

8. (a) Explain the following terms – 10
(i) Locality of reference (See Unit-III, Page 151, Q.70)
(ii) Inverted page table. (See Unit-III, Page 140, Q.56)

(b) Explain demand paging with example. 10
(See Unit-III, Page 176, Q.81)

Unit-V

9. (a) What are the design issues of distributed operating systems ?
(See Unit-V, Page 245, Q.15)

(b) What is the difference between a virus and a worm ? How do they each reproduce ? ** 10

10. (a) Compare remote procedure calls and remote evaluation on the basis of flexibility, efficiency and security. ** 10

(b) Discuss RRA protocol. ** 10

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2012
OPERATING SYSTEM
(CS-502)

Note : Total number of questions is 10. Attempt *one* question (including all parts) from each Unit. Assume missing data, if any, suitably.

Unit-I

1. (a) What is meant by operating systems ? Explain various types of operating systems in detail. (See Unit-I, Page 20, Q.31)

(b) Compare the following –
(i) Spooling and Buffering (See Unit-I, Page 29, Q.48)
(ii) Hard-Real time systems and Soft-real time systems.
(See Unit-I, Page 11, Q.13)

Or

2. (a) Explain various services provided by an operating system. Explain how each provides convenience to the users. (See Unit-I, Page 22, Q.36) Explain also in which cases it would be impossible for user level programs to provide these services.

(b) Explain the following –
(i) Spooling and Buffering (See Unit-I, Page 29, Q.48)
(ii) Multitasking and multiprogramming. (See Unit-I, Page 18, Q.26)

Unit-II

3. (a) What are various file allocation methods ? Explain Linked allocation method in detail. (See Unit-II, Page 44, Q.15)

(b) Define following –
(i) Boot block (See Unit-I, Page 29, Q.47)
(ii) Sector sparing ** (iii) Levels of RAID ** (iv) Device Driver **

Or

4. (a) The queue of pending requests, in FIFO order, is –
86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130
Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling
(i) FCFS (ii) SSTF (iii) SCAN (iv) LOOK (v) C-SCAN.
(See Unit-II, Page 77, Prob.2)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) Explain in detail about various ways of free space management.
(See Unit-II, Page 43, Q.14)

Unit-III

5. (a) Consider the following set of processes, with the length of the CPU-burst time given in milliseconds –

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

The processes are asumed to have arrived in the order P_1, P_2, P_3, P_4. P_5, all at time 0.

(i) Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a non preemptive priority and RR (quantum = 1) scheduling.

(ii) What is the turnaround time of each process for each of the scheduling algorithms ?

(iii) What is the waiting time of each process for each of the scheduling algorithms ?

(iv) Which of the scheduling in part a results in the minimal average waiting time ?
(See Unit-III, Page 114, Prob.8)

(b) What are the conditions necessary to hold for deadlock occur ?
(See Unit-IV, Page 211, Q.41)

(c) What is a semaphore ? (See Unit-IV, Page 196, Q.19)

(d) What are preemptive and non-preemptive scheduling ?
(See Unit-III, Page 97, Q.19)

Or

6. (a) What is critical section problem and explain two process solutions and multiple process solutions ? (See Unit-IV, Page 198, Q.21)

(b) Consider the following snapshot of a system –

Process	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the banker's algorithm –

(i) What is the content of the matrix Need ? Is the system in a safe state ?

(ii) If a request from process P1 arrives for (0, 4, 2, 0), can the request be granted immediately.
(See Unit-IV, Page 222, Prob.3)

Unit-IV

7. (a) Consider the following page reference string –
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
How many page faults would occur for the following replacement algorithms for three and four frames ? Remember all frames are initially empty, so your first unique pages will all cost one fault each.
(i) LRU replacement (ii) FIFO replacement (iii) Optimal replacement.
(See Unit-III, Page 170, Prob.28)

(b) Describe internal and external fragmentation with illustrative examples.
(See Unit-III, Page 132, Q.44)

(c) What are the major problems to implement demand paging ?
(See Unit-III, Page 177, Q.83)

Or

8. (a) Explain the following –
(i) MFT (ii) Paging (iii) Segmentation (iv) TLB hit/miss.
(See Unit-III, Page 144, Q.63)

(b) Define TLB with diagram. (See Unit-III, Page 142, Q.59)

(c) Write about Thrashing. Exactly when it occurs with diagram. **

Unit-V

9. (a) Explain the concept of Parallel Processing. **

(b) What do you mean by distributed operating system ? Discuss the design issues for a distributed operating system.
(See Unit-V, Page 247, Q.16)

Or

10. Write short notes on following –
(i) RMI ** (ii) Distributed shared memory **
(iii) Parallel Processing. **

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2013
OPERATING SYSTEM
(CS-502)

Note : (i) Attempt any one question from each unit.
(ii) All questions carry equal marks.

Unit-I

1. (a) Discuss multiprogramming versus single user systems in terms of throughput and CPU utilization. (See Unit-I, Page 18, Q.27) 7

(b) Compare and contrast – 7

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(i) Multiprogramming, batch and time sharing system.
(See Unit-I, Page 18, Q.28)

(ii) Network and distributed operating system.
(See Unit-V, Page 237, Q.9)

Or

2. (a) What is meant by a system call. How it can be used ? How does an application program use these calls during execution ? How is all this related to the compilation process ? (See Unit-I, Page 26, Q.45) 6

(b) Explain the function of operating system. (See Unit-I, Page 4, Q.5) 4

(c) Explain various steps involved in booting. (See Unit-I, Page 29, Q.47) 4

Unit-II

3. (a) Describe various space allocation strategies with their merits/demerits.
(See Unit-II, Page 45, Q.15) 7

(b) Suppose that a disk drive has 5000 cylinders numbered from 0 to 4999. The drive is currently serving a request at cylinder 143 and previous request was at cylinder 125. The queue of pending request in FIFO order is –
86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130 starting from the current head position. What is the total distance (in cylinder) that the disk arm moves to satisfy all the pending requests for each of the following disk scheduling algorithms – 7
(i) FCFS (ii) LOOK. (See Unit-II, Page 77, Prob.2)

Or

4. (a) Describe first fit, best fit and worst fit strategies for disk space allocations with their merits and demerits. (See Unit-III, Page 133, Q.48) 7

(b) Explain the concept of block. How does the block size affects the system performance in terms of I/O speeds and disk space utilization. 7

Unit-III

5. (a) Assume that the following jobs are to be executed on one processor.

Job	*Execution Time*	*Arrival Time*	*Priority*
0	80	0	2
1	25	10	4
2	15	20	3
3	20	30	4
4	45	40	1

Using shortest job first, priority and shortest remaining time first scheduling. Draw Gantt chart and calculate average waiting and turn around time.
(See Unit-III, Page 111, Prob.5) 7

(b) What are different process scheduling levels ? How do they interact with each other.
(See Unit-III, Page 86, Q.7) 4

(c) Explain preemptive scheduling. (See Unit-III, Page 97, Q.19) 3

Or

6. (a) Consider the following snapshot of a system – 7

	Allocation				*Maximum*				*Available*			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using bankers algorithms – 7
(i) What is the content of matrix need ?
(ii) Is the system in safe state ?
(See Unit-IV, Page 226, Prob.4)

(b) Discuss round robin scheduling policy with its merits and demerits. What is the impact of the quantum of time slice on the system performance ? (See Unit-III, Page 94, Q.15) 7

Unit-IV

7. (a) Explain paging and segmentation. How they are helpful in removing fragmentation ? (See Unit-III, Page 135, Q.53) 6

(b) Explain the concept of dirty bit for improving the performance during page fault. (See Unit-III, Page 146, Q.66) 4

(c) Explain the impact of page size on the overall system performance.
(See Unit-III, Page 140, Q.57) 4

Or

8. (a) Consider the main memory with capacity of 4 page frame. Assume that the pages of a process are referenced in the order as given below –
1, 3, 4, 4, 3, 2, 1, 7, 5, 6, 4, 2, 1, 2.
Which one is better FIFO or LRU and why ?
(See Unit-III, Page 164, Prob.20)

(b) Explain Belady's algorithm. (See Unit-III, Page 150, Q.68)

(c) Contrast demand paging versus working set model as page fetch policy. (See Unit-III, Page 177, Q.82)

Unit-V

9. (a) Contrast centralized and distributed operating system.
(See Unit-V, Page 238, Q.10)

(b) Explain RPC (Remote Procedure Call) How it implemented. **

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Or

10. (a) What is the difference between a worm and a virus ? How do worm spread ? How can they be prevented ? **

(b) What are salient features of UNIX ? (See Unit-V, Page 248, Q.20)

(c) How does UNIX provide file protection ? Explain. (See Unit-V, Page 250, Q.25)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2014
OPERATING SYSTEM
(CS-502)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each question are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What are the three main purposes of an operating system ? (See Unit-I, Page 4, Q.4)

(b) What is the difference between a hard real time system and a soft real time system ? (See Unit-I, Page 11, Q.13)

(c) Write the name of system component of OS. (See Unit-I, Page 4, Q.3)

(d) What are system calls ? Explain briefly about various types of system calls provided by an operating system. (See Unit-I, Page 27, Q.46)

Or

Enumerate the operating system services and functions. Also discuss how these service are provided. (See Unit-I, Page 23, Q.37)

Unit-II

2. (a) Write characteristic of contiguous file allocation. (See Unit-II, Page 48, Q.16)

(b) What are the operations possible on the file ? (See Unit-II, Page 34, Q.4)

(c) Explain the booting process. (See Unit-I, Page 29, Q.47)

(d) Suppose that head of a moving head disk with 200 tracks numbered 0 to 199 is currently serving a request at 50 track and has just finished a request at track 85. If the queue of requests is kept in FIFO order 100, 199, 56, 150, 25, 155, 70 and 85. A seek takes 6m sec per cylinder moved. How much seek time is needed for the SCAN ? (See Unit-II, Page 78, Prob.4)

Or

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

advantages and disadvantages. (See Unit-II, Page 43, Q.14)

Unit-III

3. (a) List the attributes of PCB. (See Unit-III, Page 85, Q.4)

(b) Define the term critical section. (See Unit-IV, Page 194, Q.17)

(c) Explain about semaphore. (See Unit-IV, Page 196, Q.19)

(d) Assume you have the following jobs to execute with one processor.

Process	*Arrival Time*	*Burst Time*
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

Calculate average turn around time using SJF Preemptive scheduling algorithm. (See Unit-III, Page 109, Prob.3)

Or

What is deadlock detection and recovery technique ? Discuss some deadlock detection methods and recovery handling procedures. (See Unit-IV, Page 218, Q.49)

Unit-IV

4. (a) What is dynamic relocation ? (See Unit-III, Page 128, Q.40)

(b) What is fragmentation ? (See Unit-III, Page 130, Q.40)

(c) Why paging and segmentation is combined ? (See Unit-III, Page 153, Q.74)

(d) Describe the working of page replacement algorithm. (See Unit-III, Page 147, Q.67)

Or

Consider the following page reference strings 1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9.
How many page faults would occur for LRU replacement algorithm ? Assume frame size = 4. (See Unit-III, Page 166, Prob.23)

Unit-V

5. (a) Define a distributed system. (See Unit-I, Page 19, Q.29)

(b) What is access matrix ? **

(c) Explain the goals of protection system. **

(d) What do you understand by a 'trusted system' ? Explain the passive and active threat to security of a system. **

Or

Discuss the need of synchronization in distributed operating system. Explain the working of any one clock synchronization algorithm. **

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2015
OPERATING SYSTEM
(CS-502)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What is bare machine ? **(See Unit-I, Page 3, Q.2)**
   (b) What is the purpose of system calls ? **(See Unit-I, Page 26, Q.44)**
   (c) How a time-sharing operating system differs from batch operating system ? **(See Unit-I, Page 13, Q.16)**
   (d) How many types of operating system are there ? Explain each of them. **(See Unit-I, Page 20, Q.32)**

   Or

   Discuss different structures of operating system with advantages and disadvantages. **(See Unit-I, Page 5, Q.7)**

Unit-II

2. (a) What is kernel I/O subsystem ? **
   (b) Define the term disk reliability. **(See Unit-II, Page 39, Q.10)**
   (c) Explain free space management technique. **(See Unit-II, Page 43, Q.14)**
   (d) Compare file system in Windows and Linux. **(See Unit-II, Page 66, Q.34)**

   Or

   Explain various methods of accessing file with examples. **(See Unit-II, Page 68, Q.37)**

Unit-III

3. (a) Define deadlock. **(See Unit-IV, Page 209, Q.36)**
   (b) Write the use of process control block and discuss its contents. **(See Unit-III, Page 84, Q.3)**
   (c) What is hard and soft semaphore ? **(See Unit-IV, Page 198, Q.23)**
   (d) What resources are used when a thread is created ? How do they differ from those used when a process is created ? **(See Unit-III, Page 120, Q.32)**

   Or

   Describe about how recovery from deadlock. **(See Unit-IV, Page 218, Q.49)**

Unit-IV

4. (a) What is thrashing ? **
   (b) Compare paging and segmentation. **(See Unit-III, Page 153, Q.73)**
   (c) Differentiate between external and internal fragmentation. **(See Unit-III, Page 132, Q.44)**
   (d) What advantages does segmentation offer over multiple variable partitions ? **(See Unit-III, Page 135, Q.51)**

   Or

   Explain cache memory organization. **(See Unit-III, Page 125, Q.37)**

Unit-V

5. (a) What is parallel processing ? **

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

   (b) Define distributed shared memory. **
   (c) Give the difficulties which can be encounter while implementing a distributed operating system. **(See Unit-V, Page 238, Q.11)**
   (d) Explain the major issues in implementing the remote procedure call (RPC) mechanism in distributed operating system. **

   Or

   Write a brief notes on parallel operating system. **(See Unit-V, Page 255, Q.32)**

RGPV

B.E. (Fifth Semester) EXAMINATION, June 2016
OPERATING SYSTEM
(CS/EI-502)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What is buffering and spooling ? **(See Unit-I, Page 29, Q.48)**
   (b) What are the two main functions of an operating system ? **(See Unit-I, Page 4, Q.5)**
   (c) Define the operating system architecture and services with the help of suitable example. **(See Unit-I, Page 23, Q.38)**
   (d) What do you understand by real time operating system ? How it is different from other operating system ? **(See Unit-I, Page 11, Q.13)**

   Or

   What do you understand by system call ? Explain its uses with the help of example. **(See Unit-I, Page 26, Q.45)**

Unit-II

2. (a) What is device driver ? **
   (b) What is file ? **(See Unit-II, Page 33, Q.2)**
   (c) Explain free space management technique. **(See Unit-II, Page 43, Q.14)**
   (d) Compare the merits and demerits of various disk scheduling algorithms. **(See Unit-II, Page 75, Q.48)**

   Or

   How can you provide protection to a file when it is shared among several users ? **(See Unit-II, Page 55, Q.24)**

Unit-III

3. (a) Define program, process and processors. **(See Unit-III, Page 83, Q.1)**

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

(b) What is inter processes communication ? (See Unit-IV, Page 189, Q.14)
(c) What is hard and soft semaphore ? (See Unit-IV, Page 198, Q.23)
(d) A system has two processes and three identical resources. Each process needs a maximum of two resources to complete. Is deadlock possible ? Explain your answer. (See Unit-IV, Page 210, Q.39)

Or

Explain Banker's algorithm for deadlock avoidance with examples. (See Unit-IV, Page 215, Q.47)

Unit-IV

4. (a) Define logical and physical address space. (See Unit-III, Page 127, Q.39)
(b) Define demand paging. (See Unit-III, Page 176, Q.81)
(c) What is the difference between global and local page replacement policy? (See Unit-III, Page 146, Q.65)
(d) Write the necessary step taken by the operating system when a page fault occurs. (See Unit-III, Page 144, Q.64)

Or

What advantages does segmentation offer over multiple variable partitions ? (See Unit-III, Page 135, Q.51)

Unit-V

5. (a) Define distributed shared memory. **
(b) What is the role of stub in RPC execution ? **
(c) Differentiate between a distributed operating system and a network operating system. (See Unit-V, Page 237, Q.9)
(d) Explain the major issues in implementing the remote procedure call (RPC) mechanism in distributed operating system. **

Or

Write features of distributed operating system. (See Unit-V, Page 245, Q.15)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2016
OPERATING SYSTEM
(CS-502)

Note : (i) Attempt any five questions out of eight questions.
(ii) All questions carry equal marks.

1. (a) What is operating system ? Define the essential properties of the following types of operating system –
(i) Batch (ii) Time sharing
(See Unit-I, Page 13, Q.17)
(b) Describe the differences among short term, medium term and long term scheduling. (See Unit-III, Page 86, Q.7)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

2. (a) Discuss various file allocation and access methods. Compare their advantages and disadvantages. (See Unit-II, Page 69, Q.38)
(b) Describe how a file directory system can be organized into one-level, two-level and tree-structure directories. (See Unit-II, Page 50, Q.20)

3. (a) Consider the following set of processes –

Process	Processing Time
A	3
B	5
C	2
D	5
E	·5

Develop a Gantt-chart and calculate the average waiting time using –
(i) FCFS (ii) SJF (iii) Round robin (q = 1)
(See Unit-III, Page 106, Prob.1)
(b) What is thread ? What resources are used when a thread is created ? How do they differ from those used when a process is created ? (See Unit-III, Page 120, Q.32)

4. (a) Briefly explain the following –
(i) Mutual exclusion (See Unit-IV, Page 188, Q.11)
(ii) Critical section problem (See Unit-IV, Page 194, Q.17)
(b) Write a semaphore solution for dining philosopher's problem. (See Unit-IV, Page 206, Q.30)

5. (a) What is deadlock ? What are the four necessary conditions for a deadlock to occur ? (See Unit-IV, Page 211, Q.42)
(b) Consider the following snap shot of a system –

	Allocation			*Request*			*Available*		
	A	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*C*
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

Answer the following questions using banker's algorithm –
(i) What is the content of the matrix need ?
(ii) If a request from process P_1 arrives for (1, 0, 2) can the request be granted immediately ?
(iii) Is the system in a safe state ?
(See Unit-IV, Page 220, Prob.1)

6. (a) Consider the following page reference string –
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults occur for the following replacement algorithm assuming three frames ?
(i) FIFO replacement
(ii) LRU replacement
(iii) Optimal replacement.
(See Unit-III, Page 170, Prob.28)

(b) What is meant by thrashing ? Explain various causes of thrashing. **

7. (a) Differentiate among the following –
(i) Physical address and logical address **(See Unit-III, Page 127, Q.39)**
(ii) Paging and segmentation **(See Unit-III, Page 153, Q.73)**

(b) What is distributed system ? Discuss the advantages of distributed systems. **(See Unit-I, Page 19, Q.30)**

8. Write short notes on the following (any four) –
(a) Worms and viruses **
(b) Key features of windows file system **(See Unit-II, Page 60, Q.28)**
(c) Distributed shared memory **
(d) Parallel operating system **(See Unit-V, Page 255, Q.32)**
(e) RPC. **

RGPV B.E. (Fourth Semester) EXAMINATION, June 2017
Choice Based Credit System (CBCS)
OPERATING SYSTEM
(IT-226)

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) What is context switching ? Discuss different type of scheduler. **(See Unit-III, Page 88, Q.8)**

(b) Define process states. Draw the diagram of PCB. **(See Unit-III, Page 88, Q.9)**

2. Write about FCFS scheduling and Round Robin scheduling which one is best in which condition. Justify your answer. **(See Unit-III, Page 95, Q.16)**

3. (a) State and explain critical section problem. **(See Unit-IV, Page 194, Q.17)**

(b) Discuss any one classical problem of synchronization. **(See Unit-IV, Page 207, Q.34)**

4. (a) Explain the resource-allocation graph algorithm for deadlock detection with relevant diagrams. **(See Unit-IV, Page 214, Q.46)**

(b) Discuss memory management techniques. **(See Unit-III, Page 128, Q.41)**

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

5. (a) Explain how logical memory address are translated into physical memory address in segmented memory management system. **(See Unit-III, Page 155, Q.76)**

(b) What are the advantages and disadvantages of contiguous and non contiguous memory allocation ? **(See Unit-III, Page 133, Q.47)**

6. (a) What is thrashing ? **
Discuss any one page replacement algorithm. **(See Unit-III, Page 147, Q.67)**

(b) Discuss the difference between demand paging and demand segmentation. **(See Unit-III, Page 178, Q.85)**

7. (a) Discuss FCFS scheduling with example. Also discuss the advantages of FCFS. **(See Unit-II, Page 76, Q.49)**

(b) Write short notes on –
(i) FAT (ii) I-node **(See Unit-II, Page 68, Q.36)**

8. (a) What is locality of reference and explain its use ? What is working set ? What is it used for ? **(See Unit-III, Page 152, Q.72)**

(b) Explan virtual memory. **(See Unit-III, Page 175, Q.80)**

RGPV IT-4003 (CBGS)
B.E. (Fourth Semester) EXAMINATION, May 2018
Choice Based Grading System (CBGS)
OPERATING SYSTEM

Note : (i) Attempt any five questions out of eight.
(ii) All questions carry equal marks.

1. (a) Explain why 'wait' and 'signal' operations on a semaphore should be executed automatically. **(See Unit-IV, Page 200, Q.25)**

(b) What are threads ? What resources are used when a thread is created ? **(See Unit-III, Page 120, Q.32)**

2. With reference to the following set of processes, determine average waiting time and average turnaround time. Using the following scheduling algorithms –
(i) Shortest remaining time first
(ii) Priority based (Preemptive).

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus

Process	Arrival Time	CPU Burst	Priority
P_1	0	24	5
P_2	3	7	3
P_3	5	6	2
P_4	10	10	1

(See Unit-III, Page 111, Prob.6)

3. Prove the Belady's Anomaly in FIFO page replacement algorithm, assuming that the number of frames is increased from 3 to 4; For the page reference strings
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
Assume all frames are empty initially. (See Unit-III, Page 165, Prob.21)

4. (a) What is the difference between non-preemptive and preemptive scheduling ? Explain why it is not likely to use strict non-preemptive scheduling in a computer system ? (See Unit-III, Page 97, Q.20)
   (b) Write Banker's Algorithm. (See Unit-IV, Page 215, Q.47)

5. (a) Write is the cause of Thrashing ? How does the system detect thrashing ? **
   (b) If the average page fault service time of 25 ms and a memory access time of 100 ns. Calculate the effective access time.
(See Unit-III, Page 172, Prob.30)

6. The request tracks in the order received are 55, 58, 39, 18, 90, 160, 150, 38, 98. Apply the following disk scheduling algorithms starting tracks at 100.
   (i) SSTF
   (ii) C-SCAN.
(See Unit-II, Page 81, Prob.6)

7. (a) Explain various file access methods. (See Unit-II, Page 68, Q.37)
   (b) Explain any one deadlock detection methods.
(See Unit-IV, Page 218, Q.49)

8. Write short notes on any four –
   (a) i-node (See Unit-II, Page 67, Q.35)
   (b) TLB (See Unit-III, Page 142, Q.59)
   (c) System call (See Unit-I, Page 25, Q.43)
   (d) PCB (See Unit-III, Page 84, Q.3)
   (e) Demand paging. (See Unit-III, Page 176, Q.81)

**Now, according to new revised syllabus of R.G.P.V., it is not included in syllabus